

Politechnika Wrocławskiego
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Techniczna (ITE)**
Specjalność: **Inżynieria Systemów Informatycznych (INS)**

**PRACA DYPLOMOWA
MAGISTERSKA**

**Analiza porównawcza jakości klasyfikatorów
danych tabelarycznych w systemach wykrywania
intruzów w sieciach komputerowych**

inż. Bartosz Błyszcz

Opiekun pracy
dr inż. Tomasz Babczyński

Słowa kluczowe: 3-6 słów

Streszczenie

Wykaz skrótów

Skrót	Nazwa angielska	Nazwa polska
GA	<i>Genetic Algorithm</i>	Algorytm Genetyczny
GP	<i>Genetic Programming</i>	Programowanie Genetyczne
GNB	<i>Gaussian Naive Bayes</i>	Naiwny Klasyfikator Bayesa wykorzystujący rozkład Gaussa
ANN	<i>Artificial Neural Network</i>	Sztuczna sieć neuronowa
CNN	<i>Convolutional Neural Network</i>	Konwolucyjna sieć neuronowa
ML	<i>Machine Learning</i>	Uczenie maszynowe
AI	<i>Artificial Intelligence</i>	Sztuczna Inteligencja
IDS	<i>Intrusion Detection System</i>	System Wykrywania Intruzów
SVM	<i>Support Vector Machine</i>	Maszyna Wektorów Nośnych
AUC	<i>Area Under Roc Curve</i>	Przestrzeń pod krzywą ROC
LCDPs	<i>Low-code Development Platforms</i>	Platforma Low-code
BI	<i>Business Intelligence</i>	Narzędzia biznesowe do przekształcania danych
CDN	<i>Content Delivery Network</i>	Sieć dostarczania zawartości
MLP	<i>Multilayer Perceptron network</i>	Sieć wielowarstwowa perceptronowa
Azure ML	<i>Azure Machine Learning Studio</i>	
GAGNB	<i>Gaussian Naive Base - with GA</i>	
HTTPS	<i>Hyper Text Transfer Protocol Secure</i>	Protokół służący do komunikacji w sieci internetowej
REST	<i>Representative State Transfer</i>	Rozwiązanie architektoniczne służące do komunikacji w sieci internetowej

Spis treści

1. Wstęp	7
1.1. Wprowadzenie i uzasadnienie tematu pracy	7
1.2. Cel i zakres pracy	8
2. Podsumowanie pracy inżynierskiej	9
2.1. Algorytm genetyczny	9
2.2. Klasyfikator Naiwny Bayesa	10
2.3. Systemy wykrywania intruzów	11
2.4. Autorski algorytm opracowany w ramach pracy inżynierskiej	11
3. Sztuczna inteligencja	13
3.1. Uczenie maszynowe	13
3.1.1. Uczenie nienadzorowane	14
3.1.2. Uczenie nadzorowane	14
3.1.3. Uczenie częściowo nadzorowane	15
3.1.4. Uczenie przez wzmacnianie	15
3.2. Sztuczna sieć neuronowa	16
3.3. Głębokie uczenie	18
4. Klasifikacja danych	20
4.1. Metryki jakościowe	20
4.1.1. Dokładność	21
4.1.2. Precyzaja	21
4.1.3. Czułość	21
4.1.4. F1	21
4.1.5. AUC	21
5. Platformy low-code/no-code	22
5.1. Microsoft PowerApps	23
5.2. Amazon QuickSight	24
5.3. Google AppSheet	24
5.4. Microsoft Azure	24
5.5. Infrastruktura	26
5.6. Machine Learning Studio	27
6. Opis doświadczenia	28
6.1. Założenie techniczne	29
6.2. Dane	29
6.3. Programistyczne środowisko badawcze	29

6.4.	Algorytmy wykorzystane w doświadczeniu.....	31
6.4.1.	Two-Class Support Vector Machine	31
6.4.2.	Two-Class Boosted Decision Tree	32
6.4.3.	Two-Class Decision Forest	33
6.4.4.	Two-class Neural Network	34
6.4.5.	Two-Class Average Perceptron.....	35
6.4.6.	Gausian Naive Bayes - with GA.....	36
6.4.7.	DANet.....	37
7.	Przebieg eksperymentu	38
7.1.	Metodologia badawcza	38
7.2.	Przygotowanie platformy badawczej.....	39
7.2.1.	Przygotowanie danych	39
7.2.2.	Trenowanie oraz testowanie algorytmów	40
7.2.3.	Utworzenie tabeli porównawczej.....	41
7.3.	Weryfikacja potoku.....	42
7.4.	Próba badawcza	44
7.4.1.	Wyniki dopasowania.....	46
7.4.2.	Wyniki precyzji.....	47
7.4.3.	Wyniki czułości.....	48
7.4.4.	Wyniki f1	49
7.4.5.	Wyniki AUC	50
7.5.	Analiza wyników	51
7.6.	Wnioski.....	52
8.	Perspektywy rozwoju.....	53

1. Wstęp

1.1. Wprowadzenie i uzasadnienie tematu pracy

Klasyfikacja danych tabelarycznych jest trudnym zagadnieniem do analizy, z powodu powszechnego występowania bardzo dużej ilości nieuporządkowanych danych, które zawierają wiele cech. To proces organizowania danych w tabeli w celu ich łatwiejszej analizy, interpretacji czy dalszego przetwarzania. Podczas takiej kategoryzacji danych ważne jest właściwe dobranie algorytmu, ze względu na typ danych. Przykładowo zbiór danych tekstowych można klasyfikować za pomocą jednokierunkowej sieci neuronowej, a obrazy za pomocą sieci konwolucyjnej.

Obecnie istnieje wiele różnych algorytmów do klasyfikacji danych tabelarycznych. Jednymi z popularniejszych są: regresja logistyczna (*ang. logistic regression*), drzewo decyzyjne (*ang. decision tree*), las losowy (*ang. random forest*), maszyna wektorów nośnych (*ang. support vector machine*), naiwny klasyfikator Bayesowski (*ang. Naive Bayes classifier*). Przy wykorzystaniu tych algorytmów do kategoryzacji, ważne jest właściwe wybranie algorytmu, czyli rozpoznanie z jakimi danymi mamy do czynienia oraz porównanie wyników klasyfikacji, w celu wybrania najlepszego dopasowania.

Dane tabelaryczne występują w każdej dziedzinie, duże zestawy danych można spotkać w medycynie, nauce czy w finansach. Rosnąca liczba danych oraz ich zmienna struktura wymaga opracowywania coraz lepszych algorytmów klasyfikacji. Jednakże wyjątkowość danych sprawia, że trudno opracować uniwersalny algorytm klasyfikacji. Wiele rozwiązań jest tworzonych dla konkretnej struktury danych, co powoduje niemożność ich wykorzystania dla innych danych.

Przy rosnącej liczbie danych do analizy, rozwijają się metody ułatwiające ich klasyfikację. Coraz częściej wykorzystuje się rozwiązania z zakresu sztucznej inteligencji czy obliczeń chmurowych. Jednym z przykładów jest aplikacja *Machine Learning Studio*, dostarczana przez *Microsoft Azure*. Zawiera ona zestaw narzędzi, umożliwiających łatwiejsze kategoryzowanie danych czy tworzenie algorytmów klasyfikacji i ich porównywanie. Użycie chmury pozwala na wykorzystanie mocy obliczeniowej sklasterowanych jednostek wirtualnych do wykonywania obliczeń na odpowiednich maszynach wirtualnych czy do budowania skomplikowanych zautomatyzowanych procesów złożonych z wielu zadań (*ang. pipeline*). Natomiast wykorzystanie sztucznej inteligencji pozwala na wprowadzenie elementu uczenia się w celu lepszego rozpoznawania danych.

Zastosowanie tych narzędzi umożliwia automatyzację procesu badawczego, porównanie wyników działania różnych algorytmów oraz znaczne przyspieszenie badań. Ma to znaczenie przy rosnącym zapotrzebowaniu na analizę dużych zestawów danych.

W dobie rozwijających się hurtowni danych oraz sztucznej inteligencji coraz więcej firm przetrzymuje ogromne zbiory danych w sieci komputerowej. Dane takie mogą być poufne

bądź o znaczeniu strategicznym. Wyciek takich danych może mieć negatywne konsekwencje dla właścicieli danych bądź osób, których dane są przechowywane. Przykładem może być dostęp do prywatnych kont bankowych albo danych medycznych. Dlatego też dane te są zabezpieczane m.in. przy użyciu kryptografii. Pozwala to na zaszyfrowanie przesyłu danych poufnych jak i samych przesyłanych danych. Jednakże osobom planującym kradzież konkretnych danych dużo bardziej zależy na uzyskaniu nieautoryzowanego dostępu do komputerów mogących posiadać dostęp do danych wrażliwych. Dzieje się tak, ponieważ dostęp do urządzeń, pozwala na wgląd nie tylko do konkretnych danych ale i do całej sieci intranetowej np. banku. Dostęp taki może zostać również wykorzystany do np. wgrania szkodliwego oprogramowania umożliwiającej założenie „tylnej furtki” (ang. *backdoor*). W celu ograniczenia możliwych nieautoryzowanych dostępów z zewnątrz powstało oprogramowanie *IDS, IPS* (ang. *Intrusion Detection system, Intrusion Prevent System*). Systemy do ostrzegania i przeciwdziałania atakom na sieć komputerową [1].

Dane sieciowe są bardzo złożone oraz posiadają dużo cech, dlatego do ich analizy można wykorzystać algorytmy uczenia maszynowego pozwalające na wykrycie nietypowego ruchu sieciowego. Klasyczne systemy IDS bazują głównie na regułach wykluczających konkretny ruch sieciowy. Przewagą uczenia maszynowego jest to, że może wykrywać anomalie niewchodzące w skład reguł bezpieczeństwa w sieci. Może to umożliwić przy niewielkim koszcie wytrenowania sieci wczesne i szybsze wykrywanie potencjalnych ataków niż w przypadku klasycznych systemów IDS opartych o reguły.

1.2. Cel i zakres pracy

Celem niniejszej pracy dyplomowej jest ocena jakości opracowanego w pracy inżynierskiej autorskiego sposobu klasyfikacji danych tabelarycznych, wykorzystującego algorytm genetyczny oraz Klasyfikator Naiwny Bayesa. W tym celu dokonano analizy porównawczej rozwiązania wraz z algorytmami dostępnymi w aplikacji *Machine Learning Studio*. Algorytmy opisano w Podrozdziale 6.4.

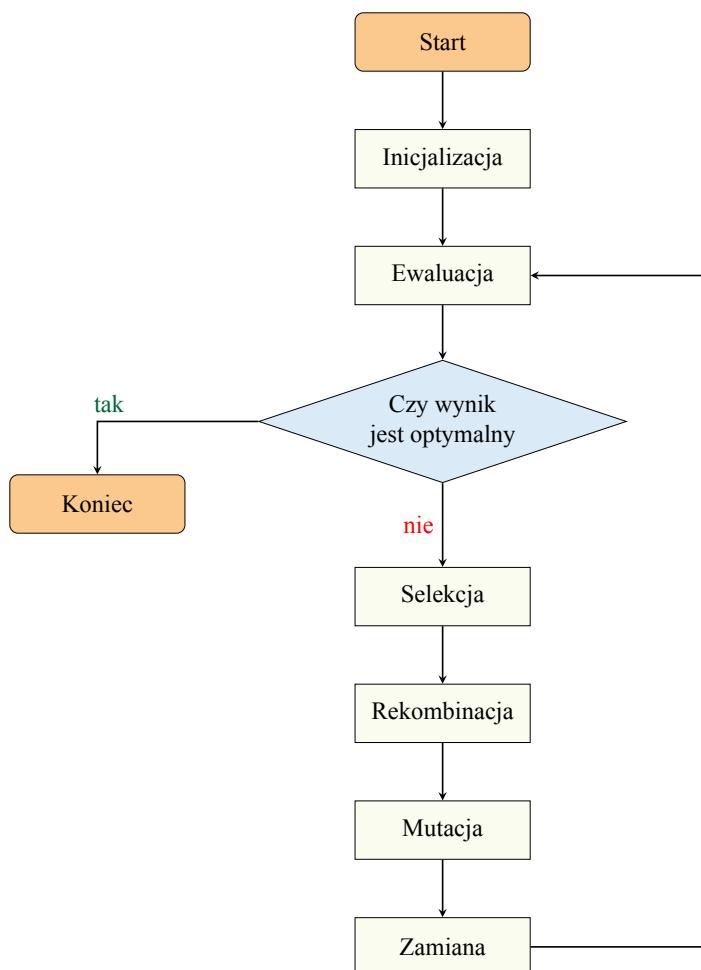
Praca składa się z 2 części. W części teoretycznej (Rozdziały 2 - 5) dokonano przeglądu dostępnych rozwiązań chmurowych (podejścia low-code/no-code). Oprócz tego opisano zagadnienia związane ze sztuczną inteligencją. W części badawczej (Rozdziały 6 - 8) przygotowano programistyczne stanowisko badawcze. W tym celu scharakteryzowano metryki jakościowe i opracowano eksperyment. Wykonano analizę porównawczą i statystyczną otrzymanych wyników. Dane wykorzystane do badań pochodziły z Instytutu Cyberbezpieczeństwa, działającego przy Uniwersytecie Nowy Brunszwik.

2. Podsumowanie pracy inżynierskiej

W ramach pracy inżynierskiej opracowano autorski algorytm, wykorzystujący algorytm genetyczny oraz klasyfikator Naiwny Bayesa. Do badań wykorzystano dane uzyskane z Instytutu Cyberbezpieczeństwa, działającego przy Uniwersytecie Nowy Brunszwik. Dane użyto też w niniejszej pracy dyplomowej. Zbiór danych został opisany w **sekcji 6.2**.

2.1. Algorytm genetyczny

Algorytm genetyczny, który został zastosowany w autorskim rozwiążaniu, to algorytm, który przeszukuje przestrzeń alternatywnych rozwiązań dla danego problemu, by znaleźć najlepszy wynik [2]. Działanie algorytmu genetycznego jest inspirowane m.in. ewolucją biologiczną, stąd też pochodzą nazwy kolejnych kroków, które zostały przedstawione na **rysunku 2.1**.



Rys. 2.1. Schemat algorytmu genetycznego
Źródło: Opracowanie własne na podstawie: [3, 2, 1]

Poniżej opisano przebieg algorytmu:

1. **Inicjalizacja** - metoda inicjująca populację losowych osobników, wykorzystywanych podczas obliczeń;
2. **Ewaluacja** - etap sprawdzenia jakości wygenerowanych danych;
3. **Selekcja** - część algorytmu skupiająca się na wybraniu n najlepszych rozwiązań, które przejdą do następnej populacji;
4. **Rekombinacja** - element tworzący nową populację na bazie poprzedniej populacji. Wykorzystuje do tego mechanizm krzyżowania rodziców;
5. **Mutacja** - metoda wprowadzająca zmianę w n losowych miejsc w genomie;
6. **Zamiana** - miejsce wymiany starej populacji na nową [3, 2, 1].

2.2. Klasyfikator Naiwny Bayesa

Autorski algorytm w etapie ewaluacji wykorzystuje naiwny klasyfikator Bayesa oparty o rozkład normalny Gaussa (*ang. Gaussian Naive Bayes, GNB*). Równanie klasyfikatora oparte jest o twierdzenie Bayesa. Opisuje ono prawdopodobieństwo wystąpienia zdarzenia na podstawie znajomości warunków zdarzenia [4]. Twierdzenie zostało przedstawione na **równaniu 2.1**.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \quad (2.1)$$

gdzie:

- H oraz X są różnymi zdarzeniami;
- $P(X) \neq 0$;
- $P(H|X)$ - prawdopodobieństwo wystąpienia zdarzenia H jeśli zdarzenie X jest prawdziwe;
- $P(X|H)$ - prawdopodobieństwo wystąpienia zdarzenia X jeśli zdarzenie H jest prawdziwe;
- $P(H)$ oraz $P(X)$ to prawdopodobieństwa zaobserwowane, bez żadnych warunków [5].

Równanie wykorzystujące rozkład normalny Gaussa zostało przedstawione na **równaniu 2.2**. W kodzie wykorzystano bibliotekę scikit-learn [6]. Posiada ona implementację GNB.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x_i-\mu_k)^2}{2\sigma_k^2}} \quad (2.2)$$

gdzie:

- σ_k^2 - wariancja danych w kolumnie,
- σ_k - odchylenie standardowe danych w kolumnie,
- μ - średnia wartość w kolumnie [5].

2.3. Systemy wykrywania intruzów

2.4. Autorski algorytm opracowany w ramach pracy inżynierskiej

Celem algorytmu była redukcja wymiarowości danych tabelarycznych, by zwiększyć jakość klasyfikacji danych. Uzyskano to poprzez wyznaczenie najistotniejszych cech zbioru. Algorytm genetyczny zastosowano w celu przygotowania zbioru prawdopodobnie istotnych kolumn poprzez oznaczenie ich pozycji za pomocą ciągu cyfr 0 i 1. Kolumny oznaczone cyfrą 1 były brane pod uwagę w procesie ewaluacji za pomocą klasyfikatora Naiwnego Bayesa. Cały proces trwał maksymalnie 1000 iteracji lub do momentu uzyskania minimum 90% dopasowania. Wynikiem działania autorskiego algorytmu jest zbiór kolumn istotnych w procesie klasyfikowania. W celu sprawdzenia jakości własnego rozwiązania wykorzystano następujące metody statystyczne:

- **metoda ANOVA** - analiza wariancji,
- **współczynnik korelacji Pearsona** - współczynnik określający zależność liniową pomiędzy zmiennymi losowymi,
- **współczynnik korelacji rang Spearmana** - współczynnik korelacji rang Pearsona, dla danych po ustaleniu rang.

Analiza wykazała, że wyniki uzyskane za pomocą metod statystycznych były gorsze niż autorskiego algorytmu. Rezultaty badań przedstawia **tabela 2.1** oraz **wykres 2.2**.

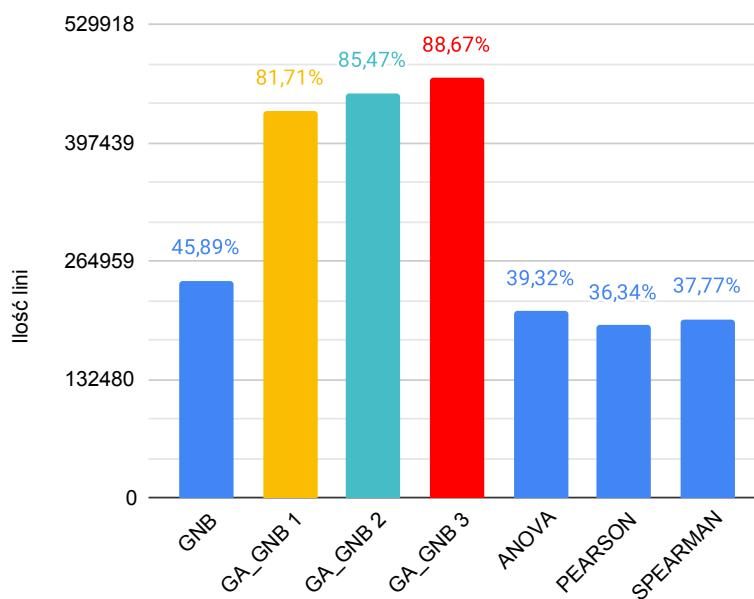
Tabela 2.1. Klasyfikacja zbioru danych: Monday-WorkingHours
Źródło: [1]

Klasyfikacja: Monday-WorkingHours							
	Podstawowa	Zoptymalizowana GA			Zoptymalizowana Statystycznie		
		1	2	3	ANOVA	PEARSON	SPEARMAN
Rozmiar danych [MB]	347,19	197,60	181,43	189,51		197,60	
Ilość linii [-]	529 918						
Czas operacji [s]	2,68	1,84	1,77	2,23	1,89	1,83	1,91
Dokładność [%]	45,89	81,71	85,47	88,67	39,32	36,34	37,77
Precyza [%]	100						
Czulość [%]	45,89	81,71	85,47	88,67	39,32	36,34	37,77
F1 [%]	62,91	89,94	92,16	93,99	56,45	53,31	54,83
Zużycie pamięci [MB]	1853,44	980,06	885,42	932,74		980,06	

Wykres 2.2. przedstawia wyniki dokładności. Zastosowano następujące opisy:

- **GNB** - dopasowanie danych w podstawowym zbiorze;
- **GA_GNB 1** - dopasowanie danych w pierwszej próbie wykorzystania algorytmu autora;
- **GA_GNB 2** - dopasowanie danych w drugiej próbie wykorzystania algorytmu autora;
- **GA_GNB 3** - dopasowanie danych w trzeciej próbie wykorzystania algorytmu autora;
- **ANOVA** - dopasowanie danych z wykorzystaniem cech uzyskanych metodą ANOVA;
- **PEARSON** - dopasowanie danych z wykorzystaniem cech uzyskanych metodą współczynników korelacji Pearsona;

- **SPEARMAN** - dopasowanie danych w zbiorze zmodyfikowanym o rozwiązanie uzyskane metodą współczynników korelacji rang Spearmana.

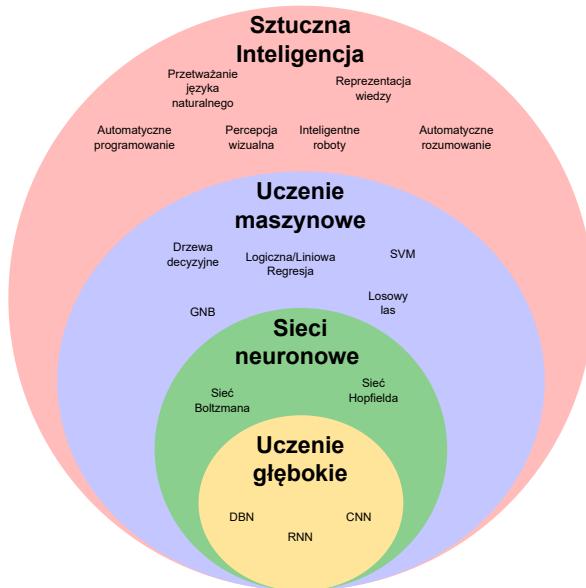


Rys. 2.2. Klasyfikacja zbioru danych: Monday-WorkingHours
Źródło: [1]

Analizując powyższe wyniki można zauważyć, że najwyższe dopasowanie uzyskał autorski algorytm a najniższe - zbiór kolumn wyznaczonych metodą współczynników korelacji Pearsona. Świadczy to o wysokiej jakości, opracowanego w pracy inżynierskiej, algorytmu.

3. Sztuczna inteligencja

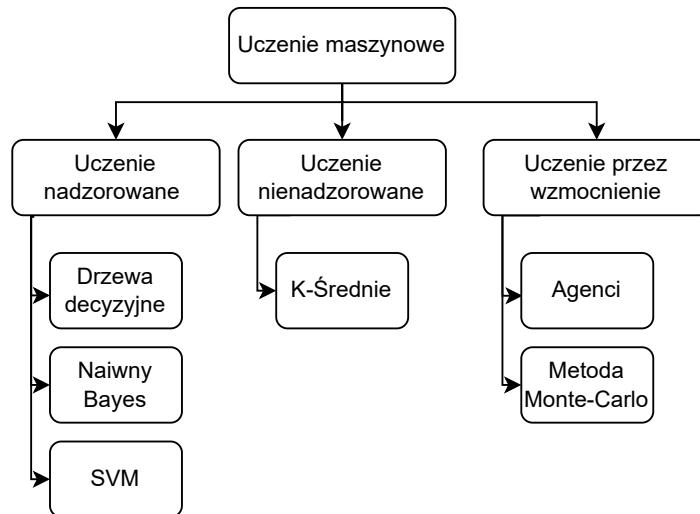
Słownik *Oxford English Dictionary* słowo „**inteligencja**” definiuje jako zdolność do rozumienia, analizy i dostosowania się do zmian [7]. Rozwój nauki i komputerów umożliwił badania nad tworem zbliżonym do możliwości ludzkiej inteligencji, czyli nad sztuczną inteligencją (*ang. Artificial Intelligence, AI*). W ostatnich latach AI dynamicznie się rozwija, znalazła zastosowanie w medycynie, finansach czy w badaniach naukowych. Wykorzystywana jest niemal na każdym etapie procesu badawczego: stawiania hipotez, budowania twierdzeń matematycznych, tworzenia i monitorowania badań, zbierania danych czy w wielu innych czynnościach [8, 9]. Wśród obszarów sztucznej inteligencji możemy wydzielić m.in. uczenie maszynowe, sieci neuronowe czy uczenie głębokie. **Rysunek 3.1** pokazuje rosnące zawężenie zagadnienia związanego z AI, gdzie uczenie głębokie jest najścisłejzym a sztuczna inteligencja najszerzym.



Rys. 3.1. Graficzne przedstawienie podziałów sztucznej inteligencji
Źródło: [10]

3.1. Uczenie maszynowe

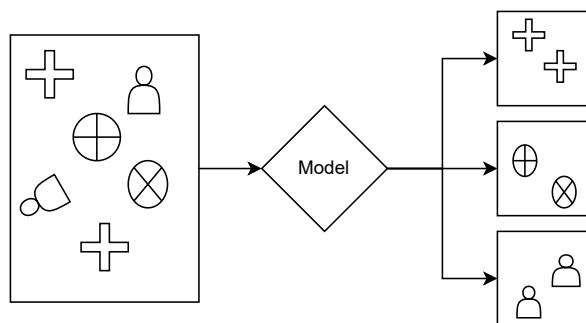
Uczenie maszynowe (*ang. Machine Learning, ML*) to dziedzina nauki nad algorytmami oraz modelami statystycznymi, które pracują na dużych zbiorach danych, ucząc się je klasyfikować. Algorytmy ML wykorzystuje się m.in. w analizie danych, tworzeniu modeli predykcyjnych, rozpoznawaniu zdjęć czy mowy. Dodatkowo nie są zaprogramowane specyficznie pod konkretne zadanie, a jedynie pod grupę zadań. Uczenie maszynowe można podzielić na 3 typy, do których należą uczenie: nadzorowane, nienadzorowane oraz ze wzmacnieniem. Podział pokazano na **rysunku 3.2**. Wykorzystanie konkretnego typu uczenia determinuje typ zadania, jaki ma być rozwiązyany [9, 10].



Rys. 3.2. Podział uczenia maszynowego
Źródło: [9]

3.1.1. Uczenie nienadzorowane

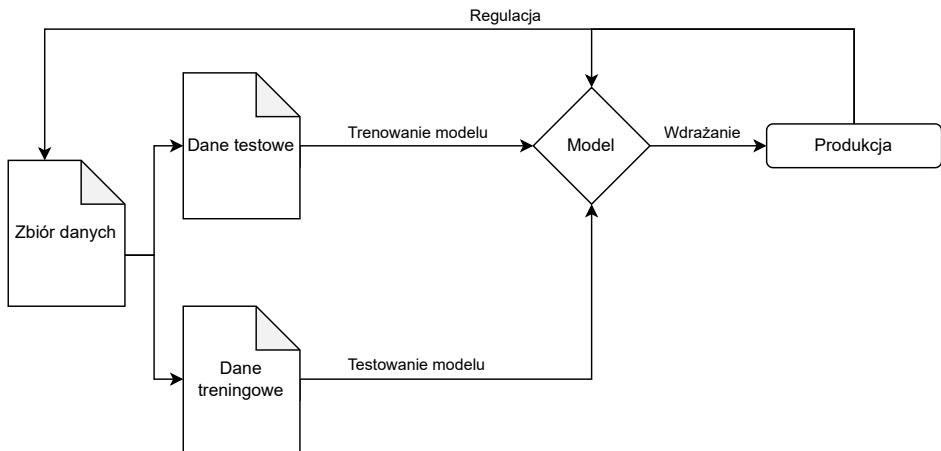
W przypadku uczenia nienadzorowanego algorytm ma sam znaleźć czy występują jakieś wzorce. W porównaniu do uczenia nadzorowanego, tu nie stosuje się zbioru oznaczonego. Metodę wykorzystuje się w pracy z danymi, których nie da się zdefiniować np. przy detekcji anomalii, szukaniu wzorców. Pozwala to m.in. na segmentację grup czy wykrywanie anormalnych zachowań np zwiększonego zużycia prądu spowodowanego uszkodzeniem urządzenia elektrycznego. Wśród algorytmów uczenia nienadzorowanego wyróżniamy dwie główne metody: K-średnie, klasteryzacja. Schemat uczenia nienadzorowanego poprzez klasteryzację jest pokazany na **rysunku 3.3** [8, 9].



Rys. 3.3. Uczenie nienadzorowane
Źródło: Opracowanie własne

3.1.2. Uczenie nadzorowane

W procesie uczenia nadzorowanego stosuje się zbiór posiadający etykiety. Model uczy się przyporządkowywać określone cechy do konkretnych kategorii. Dane wejściowe dzielone są na dane treningowe i dane testowe. Zbiór treningowy jest wykorzystywany do trenowania modelu, a zbiór testowy do sprawdzenia rezultatu. W trakcie trenowania występuje regulacja modelu. Model uczenia został przedstawiony na **rysunku 3.4** [8, 9].



Rys. 3.4. Uczenie nadzorowane

Źródło: [9]

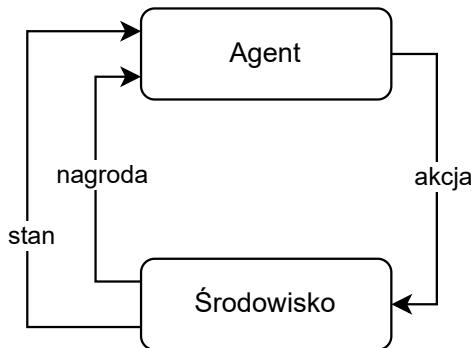
Algorytmy uczenia nadzorowanego można zastosować między innymi do weryfikacji ruchu sieciowego w celu określenia czy ruch bezpieczny, przez co można to zastosować w systemach wykrywania intruzów (*ang. Intrusion Detection System, IDS*). Algorytmy wchodzące w skład uczenia nadzorowanego to m.in. klasyfikator Naiwny Bayesa, drzewo decyzyjne, maszyna wektorów nośnych [8, 9].

3.1.3. Uczenie częściowo nadzorowane

Jednym ze szczególnych przypadków połączenia uczenia nadzorowanego i nienadzorowanego jest uczenie częściowo nadzorowane. Wykorzystywane jest w sytuacjach, kiedy istnieje zbyt duży zbiór danych, którego oznaczenie wiązałoby się z zbyt dużymi kosztami czasowymi i finansowymi. Wtedy etykietyzowana jest mała próbka danych, która służy do utworzenia modelu. Na podstawie takiego modelu oznacza się pozostały zbiór danych. Takie rozwiązanie stosuje się m.in. w bankowości, klasyfikowaniu stron internetowych poprzez wyszukiwanie treści na stronie i kategoryzowaniu ich oraz wyznaczaniu trasy GPS [11, 9].

3.1.4. Uczenie przez wzmacnianie

Uczenie przez wzmacnianie jest najbliższym ludzkiemu uczeniu się. Zbliżona jest do metody „*kija i marchewki*”. To jest uczenie poprzez nagradzanie dobrych rozwiązań, a karanie złych. W odróżnieniu od pozostałych metod uczenia maszynowego, w uczeniu przez wzmacnianie nacisk kładziony jest na przygotowanie odpowiedniego środowiska. Jako środowisko określa się zadania lub symulacje, z którymi agent wchodzi w interakcję. Schemat działania przedstawiono na **rysunku 3.5**. W tym typie uczenia nie są ważne dane, a rezultat.



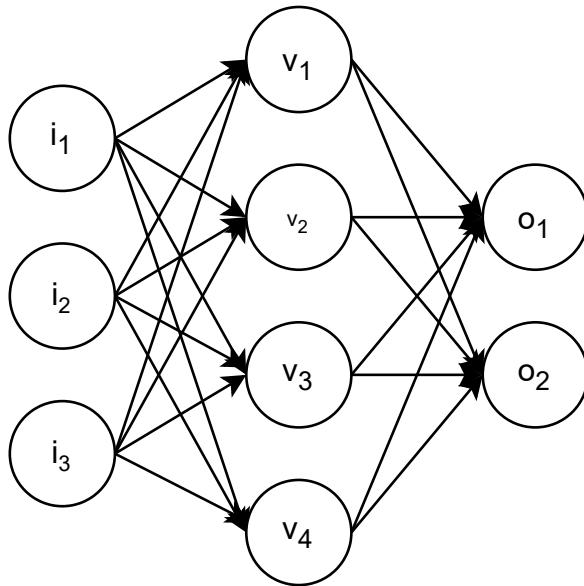
Rys. 3.5. Uczenie przez wzmocnienie
Źródło: [9]

Uczenie przez wzmocnienie jest wykorzystywane m.in. w trenowaniu pojazdów autonomicznych. Sztuczna inteligencja znajduje się np. w środowisku miejskim i zbiera dane z otoczenia. AI jest nagradzane za wybór lepszych tras do jazdy, czyli dróg asfaltowych zamiast polnych [8, 9].

3.2. Sztuczna sieć neuronowa

Ludzki mózg jest najbardziej złożonym organem znany ludziom. Składa się z wielu połączonych ze sobą komórek neuronowych, które przetwarzają równolegle wiele informacji. Ta struktura zainspirowała naukowców do zaimplementowania podobnych rozwiązań w komputerach. Przykładem tego są algorytmów, wchodzące w skład sztucznych sieci neuronowych (*ang. artificial neural network, ANN*), m.in. sieci Kohonena, sieci Hopfielda, sieci konwolucyjne. Sieci te podczas wykonywania np. klasyfikacji danych, próbują odwzorować działanie ludzkiego mózgu. Mimo tych osiągnięć symulacja ludzkiej świadomości oraz emocji wciąż jest jedynie w sferach fantazji [12].

Sztuczna sieć neuronowa to model uczenia maszynowego, który podejmuje decyzje w sposób zbliżony do ludzkiego mózgu. Budowa i działanie sieci jest opartne na działaniu ludzkich neuronów. ANN jest zbudowana z połączonych ze sobą warstw sztucznych neuronów. Sieć neuronowa posiada warstwę wejściową, warstwy ukryte i warstwę wyjściową. Każda warstwa składa się ze zbioru sztucznych neuronów. Przykładowa sieć została przedstawiona na **rysunku 3.6**.



$i_x \forall x \in [1, 2, 3]$: dane wejściowe

$v_x \forall x \in [1, 2, 3, 4]$: neurony w warstwie ukrytej

$o_x \forall x \in [1, 2]$: dane wyjściowe

Rys. 3.6. Schemat sieci neuronowej

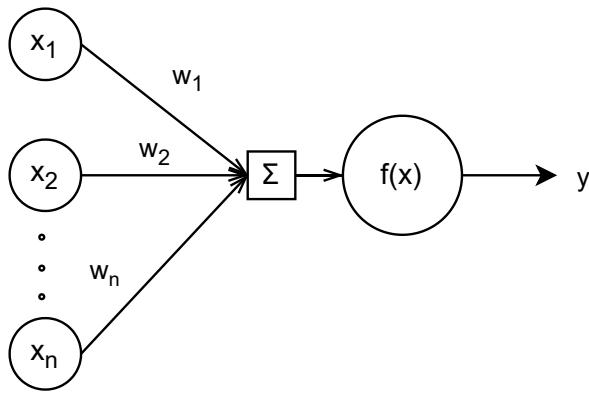
Źródło: Opracowanie własne

Warstwa wejściowa za pomocą węzłów wejściowych przyjmuje i przetwarza dane wejściowe. Dane są analizowane i przeliczane, a następnie przekazywane do warstwy ukrytej. Sieci mogą posiadać dużą ilość warstw ukrytych. Jeśli posiadają tych warstw znaczną ilość, zaczyna się je określać jako głębokie sieci neuronowe. Głębokie sieci neuronowe zostały opisane w **sekcji 3.3**. Po przejściu danych przez warstwy ukryte trafiają do warstwy końcowej, a następnie zwracany jest wynik obliczeń. W warstwie wyjściowej może być n węzłów, gdzie $n \geq 1$. W przypadku klasyfikacji binarnej (0/1) otrzymamy jeden wynik, a w przypadku klasyfikacji wieloklasowej wyników będzie więcej.

Sieć ta potrafi się dostosowywać do danych wejściowych tak, aby uzyskać odpowiedni wynik. W tym celu wykonuje wtedy proces uczenia, stosując do tego na przykład algorytm wstępnej propagacji wag. W zależności od problemu istnieje wiele różnych sieci, które można zastosować. Jednym z trudniejszych rzeczy w doborze sieci jest dobór warstw ukrytych oraz ilości neuronów, ponieważ dobór jest oparty o własną wiedzę i doświadczenie twórcy.

Sieci neuronowe możemy podzielić na wiele rodzajów. Wśród nich możemy wyróżnić między innymi:

- **perceptron** - jest to najstarszy i najprostszy przykład sieci neuronowej złożonej z jednego perceptronu (neuronu). Został przedstawiony na **rysunku 3.7**. Stosuje się go jako punkt wyjściowy do tworzenia bardziej złożonych modeli ANN opartych o warstwy złożone z perceptronów. Może on też posłużyć do klasyfikacji binarnej, w której oczekujemy jako wyniku 1 albo 0.



Σ : sumator

$f(x)$: funkcja aktywacyjna

$w_x \forall x \in [1, 2, \dots, n]$: wagi

$x_x \forall x \in [1, 2, \dots, n]$: wejścia

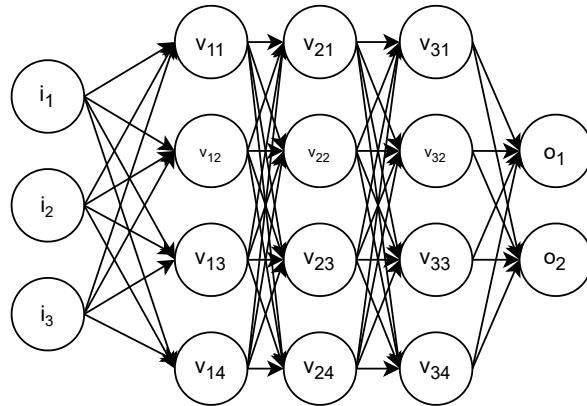
Rys. 3.7. Schemat pojedynczego neuronu - perceptronu

Źródło: Opracowanie własne

- **sieci wielowarstwowe perceptronowe** - jest to sieć złożona z wielu warstw połączonych ze sobą neuronów, najprostszy model sieci zaprezentowany na **rysunku 3.6**. Składa się z warstwy wejściowej, warstw (jednej bądź wielu) ukrytych oraz warstwy wyjściowej. W neurony w tej sieci w porównaniu do perceptronów, mają funkcję aktywacyjną sigmoidalną, ze względu na rozwiązywanie problemów nieliniowych (posiadających więcej rozwiązań niż dwa 0/1). Można je zastosować na przykład do klasyfikacji danych;
- **sieci konwolucyjne (CNN)** - są to sieci służące do rozpoznawania obrazów, nazwa wzięła się od wykonywanej na obrazie operacji konwolucji (splotu). Sieci te posiadają dodatkowe warstwy konwolucyjne oraz spłaszczania, które pozwalają zamienić reprezentację obrazu w pojedyncze wartości;
- **sieci rekurencyjne** - charakteryzują się pętlą zwrotną w warstwie ukrytej. Mogą być wykorzystane do generowania tekstu, tłumaczeń maszynowych, a także na przykład przewidywania cen rynkowych;
- **sieci samoorganizujące się** - wykorzystuje uczenie nienadzorowane oraz. Składają się jedynie z warstwy wejściowej i wyjściowej. Zaś cechą charakterystyczną jest to, że neurony określające podobne klasy znajdują się obok siebie. Sieci te mogą być wykorzystywane do podziału klientów na odpowiednie grupy bądź do wskazania, jakim klientom zaproponować karty kredytowe [13, 14].

3.3. Głębokie uczenie

Jest to podkategoria uczenia maszynowego polegająca na tworzeniu wielowarstwowych sieci neuronowych. W porównaniu do podstawowych sieci neuronowych potrzebuje ogromnych zbiorów danych do utworzenia modelu predykcyjnego. Potrzebuje również dużo więcej mocy obliczeniowej przez wzgląd na ilość warstw ukrytych, których może być dużo więcej, przykładem najprostszej sieci głębokiej jest **rysunek 3.8**.



$i_x \forall x \in [1, 2, 3]$: dane wejściowe

$v_x \forall x \in [11, 12, 13, 21, 22, 23, 31, 32, 33]$: neurony w warstwie ukrytej

$o_x \forall x \in [1, 2]$: dane wyjściowe

Rys. 3.8. Schemat prostej głębokiej sieci neuronowej
Źródło: Opracowanie własne

Warstwa wyjściowa DNN może dostarczać dane różnego formatu, może to być na przykład, tekst, liczba bądź dźwięk. Posiada również bardzo dużo zastosowań, w których skład wchodzi generowanie treści, Deepfake, analiza obrazów, wskazywanie obiektów na obrazach, projektowanie leków, czatboty. Jest to udoskonalenie podstawowych sieci neuronowych. Tak więc część typów sieci opisanych w **sekcji 3.2** będzie odnosić się do głębokich sieci neuronowych, należy do nich CNN [15].

4. Klasyfikacja danych

Człowiek od zawsze próbuje skategoryzować i uporządkować posiadaną wiedzę w zbiory pozwalające na łatwy dostęp do przechowywanych informacji na przykład w sposób tabelaryczny. Klasyfikacja to proces rozpoznawania obiektów na bazie analizy ich cech. Jest to jedna z pierwszych rzeczy, której uczą się niemowlęta: rozróżniania kształtów, kolorów czy własnych rodziców. W późniejszych latach te umiejętności są bardzo ważne, ponieważ w otaczającym świecie wiele rzeczy jest sklasyfikowanych. Np książki w katalogach bibliotecznych czy produkty spożywcze według jakości produkcji.

W uczeniu maszynowym klasyfikacja jest to metoda uczenia nadzorowanego, podczas której model próbuje przewidzieć etykietę obiektu na podstawie jego cech. Proces uczenia modelu klasyfikacji jest oparty o dwa zbiory, treningowy i testowy. Zbiór treningowy powinien być mniejszy od zbioru testowego. Po udanym procesie trenowania modelu następuje proces testowania, na podstawie którego wylicza się odpowiednie metryki pozwalające na ewaluację modelu. W pracy magisterskiej wykorzystano zbiór danych, który posiada dwa typy etykiet [0, 1], dlatego też na tej podstawie zbudowano macierz pomyłek.

4.1. Metryki jakościowe

W trakcie ewaluacji algorytmu służącego do klasyfikacji wykorzystuje się metryki bazujące na macierzy pomyłek, która została przedstawiona w **tabeli 4.1**

Tabela 4.1. Macierz pomyłek
Źródło: Opracowanie własne

		Prawdziwe wartości	
		1	0
Przewidziane wartości	1	TP	FN
	0	FP	TN

Macierz pomyłek pokazuje jakie wyniki możemy otrzymać w procesie klasyfikacji. Wyróżniamy 4 rodzaje rezultatów:

- **TP** - prawdziwie pozytywny
- **FN** - fałszywie negatywny
- **FP** - fałszywie pozytywny
- **TN** - prawdziwie negatywny

Powyższe wartości pozwalają ocenić jakość algorytmów klasyfikujących dane.

4.1.1. Dokładność

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

Stosunek wszystkich dobrze oznaczonych obiektów do liczby wszystkich prób.

4.1.2. Precyza

$$\frac{TP}{TP + FP} \quad (4.2)$$

Stosunek poprawnie wybranych obiektów klasy „I”, do wszystkich wybranych obiektów tej klasy.

4.1.3. Czułość

$$\frac{TP}{TP + FN} \quad (4.3)$$

Stosunek poprawnie sklasyfikowanych obiektów klasy „I”, do wszystkich obiektów, które powinny być w tej klasie.

4.1.4. F1

$$\frac{2 * x * y}{x + y} \quad (4.4)$$

Jest średnia harmoniczna precyzji (x) i czułości (y);

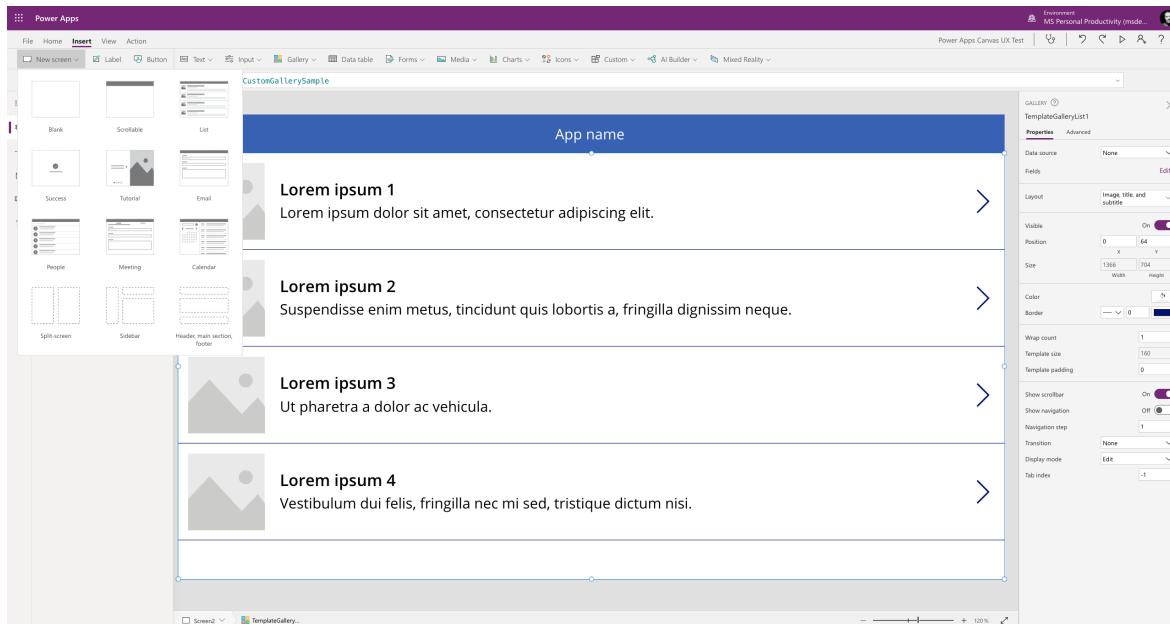
4.1.5. AUC

AUC - (*Area Under Roc Curve*) - Jest to pole pod krzywą ROC, pokazuje sprawność klasyfikatora. Im wyższa wartość AUC tym lepiej. Wynik AUC jest z zakresu $<0, 1>$:

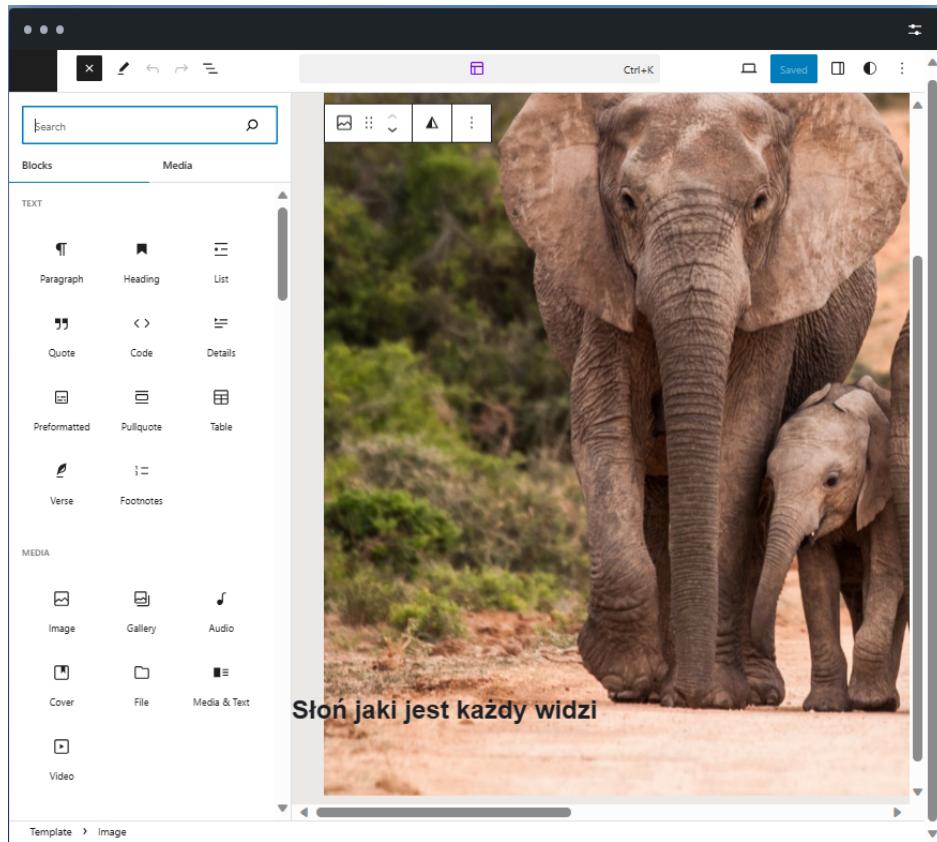
- AUC = 1 - klasyfikator idealny,
- AUC = 0,5 - klasyfikator losowy,
- AUC < 0,5 - klasyfikator gorszy niż losowy [16].

5. Platformy low-code/no-code

Low-Code oraz no-code to nowe podejście skupiające się na umożliwieniu tworzenia programów w sposób nie wymagający znajomości języka oprogramowania. Podejście to ma pozwolić osobom, które nie są programistami na tworzenie aplikacji biznesowych. Ma to zwiększyć tempo tworzenia rozwiązań biznesowych, na które zapotrzebowanie wciąż rośnie. Jednakże podejście to jest stosunkowo świeże. Jego początki można, było obserwować w codziennym życiu. Przejawiało się możliwością tworzenia stron internetowych, przy wykorzystaniu narzędzi takich jak *Wordpress*, *Joomla*, *Wix*. Narzędzia te umożliwiają w łatwy sposób utworzenie strony internetowej składającej się z tak zwanych kafelków. Umieszczone w odpowiednim miejscu *kafelki* były odpowiedzialne za jedną konkretną rzecz [17, 18, 19]. Przykład na **obrazie 5.1, 5.2.**



Rys. 5.1. PowerApps od Microsoft
Źródło: [20]



Rys. 5.2. Wordpress.com

Źródło: [21]

Coraz większą popularnością cieszą się platformy low-code (*ang. Low-code Development Platforms*) (**LCDPs**) dostarczane między innymi przez Google, Microsoft, Amazon. Pozwalają one na tworzenie wysoko skalowalnych rozwiązań przy niewielkim albo i żadnym nakładzie pracy z kodem. Ma to umożliwić osobom z niewielkim doświadczeniem w programowaniu, na szybkie wdrożenie oraz tworzenie niezawodnego oprogramowania. Twórcy platform oferują również korzystającym, zmniejszenie ilości pracy potrzebnej do wdrożenia albo rozwijania kolejnych funkcjonalności [22, 23].

LCDP udostępniane twórcom aplikacji, umożliwiają skalowalność rozwiązań tworzonych na własne potrzeby. Dodatkowo są popularne przy tworzeniu aplikacji typu „*aplikacja jako usługa*” (*ang. Software-as-a-Service*) (SaaS), które opłacane są tylko za stopień ich użycia. To podejście w konkretnych sytuacjach może się okazać dużo bardziej opłacalne niż utrzymywanie swoich rozwiązań serwerowych. Dzięki takim rozwiązaniom wiele małych firm będzie mogło pozwolić sobie na tworzenie i utrzymywanie dostosowanych rozwiązań opartych o ekosystem *Microsoft365 / Google Workspace*.

5.1. Microsoft PowerApps

Jest to platforma programistyczna umożliwiająca tworzenie niestandardowych aplikacji dla rozwiązań biznesowych. Umożliwia ona tworzenie aplikacji opartych o różnorakie źródła danych. Do których należą między innymi: SQL Server, SharePoint, Dynamics 365. Dodatkową zaletą tego rozwiązania jest tworzenie aplikacji responsywnych, działających

dobrze na wielu rodzajach urządzeń. Dodatkowo platforma ta pozwala tworzyć trzy typy aplikacji przy braku konieczności kodowania [24]

- **Kanwa** - jest to typ aplikacji oparty o model danych znajdujący się na przykład w Excelu. Aplikację tego typu tworzy się za pomocą przesuwanych kafelek, a proces przypomina po trochę robienie prezentacji przy użyciu aplikacji Powerpoint. Umożliwia to pełną dowolność w tworzonym interfejsie graficznym [25].
- **Oparte na modelu** - w ramach korzystania z usługi *Microsoft Dataverse* można wygenerować aplikacje bazujące na danym modelu danych. Dzięki czemu użytkownicy otrzymują produkt ułatwiający im analizę danych [26].
- **Karty** - są to uproszczone aplikacje, które można dodać do usługi Microsoft Teams w określonym biznesowym celu. Dużą zaletą tego rozwiązania jest możliwość korzystania ze źródeł danych. Rozwiążanie to wprowadza możliwość tworzenia aplikacji o pojedynczej odpowiedzialności biznesowej [27].

5.2. Amazon QuickSight

Jest to rozwiązanie firmy Amazon, które umożliwia firmom dostarczanie rozwiązań z zakresu analityki biznesowej (*ang. business intelligence*) (BI). Rozwiązanie to dostarcza interaktywne pulpity korzystające z jednego źródła prawdy. Dodatkowo korzystanie z interaktywnych formularzy, raportów oraz zapytań w języku naturalnym pozwala interesariuszom otrzymać możliwość korzystania z jednolitych rozwiązań opartych o różne modele danych [28].

5.3. Google AppSheet

Platforma AppSheet od firmy Google umożliwia tworzenie aplikacji mobilnych oraz desktopowych bez użycia kodu. Firma wskazuje na możliwości integracyjne z różnymi dostawcami danych, do których należą między innymi Microsoft, Dropbox. Dodatkowo posiada wbudowaną integrację z aplikacjami Google Workspace, do których należą Gmail, Sheets oraz Spaces. Platforma pozwala również na tworzenie automatycznych botów, które wykonują zadania po interakcji z bodźce zewnętrznymi bądź wewnętrznymi. Narzędzie pozwala w prosty sposób na tworzenie szybkich rozwiązań biznesowych w ekosystemie firmy Google [29].

5.4. Microsoft Azure

Platforma została oddana do użytku w 2008 roku jako Windows Azure. Usługa ta została zbudowana na modułach Windows NT. Platforma została udostępniona komercyjnie po 2010 roku, kiedy to dodano możliwość korzystania z szerszej ilości usług i języków programowania. Do usług należało między innymi udostępnienie baz danych Microsoft SQL Server opartych o .NET Framework 4, obsługę aplikacji pisanych w wielu języku (takich jak *C#, Java, PHP*), sieć dostarczania zawartości (*ang. Content Delivery Network*) (CDN).

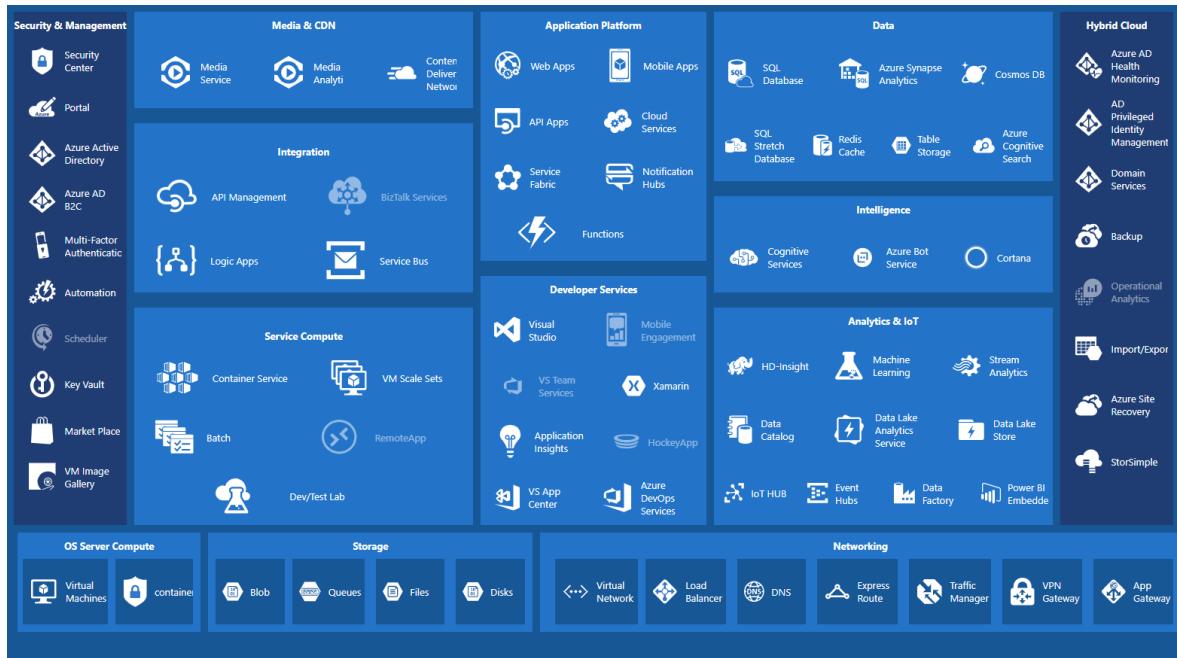
Następnym krokiem było przemianowanie platformy na Microsoft Azure oraz pójście

w kierunku infrastruktury definiowanej jako serwis (*ang. Infrastructure-as-a-Service*) (**IaaS**), oraz powolne adoptowanie usług open-source.

W kolejnej generacji Microsoft zaadoptował rozwiązania Big Data do swojej platformy, umożliwiając korzystanie z języka *R*, połaczenie do Power BI, a także umożliwienie połączenia do rozwiązań end-to-end.

W czwartej generacji platformy, Microsoft skupił się na rozwiązańach uczenia maszynowego oraz integracji z bazami danych, dzięki czemu powstało Azure Machine Learning Studio oraz Azure Machine Learning Operations (MLOps).

Obecnie platforma została wzbogacona o Kuberntesa, dzięki czemu konteneryzacja ułatwia pracę z klastrami wirtualnymi. Wirtualne klastry pozwalają na wydajniejszy i wygodniejszy sposób zarządzania aplikacjami i usługami. Dodatkowo zostało udostępnione wiele kombinacji usług takich jak: aplikacja jako usługa (*ang. Software-as-a-Service*) (SaaS), Interfejs jako usługa (*ang. Infrastructure-as-a-Service*) (IaaS), Platforma jako usługa (*ang. Platfrom-as-a-Service*) (PaaS). Microsoft uzyskał w ten sposób platformę przyjazną użytkownikowi, która umożliwia użytkownikom korzystanie z ponad 200 dostępnych usług. Dodatkowo płatność za platformę jest rozliczana tylko za zadaną przestrzeń oraz wykorzystaną moc obliczeniową [30, 31, 32].



Rys. 5.3. Schemat podziału usług MS Azure

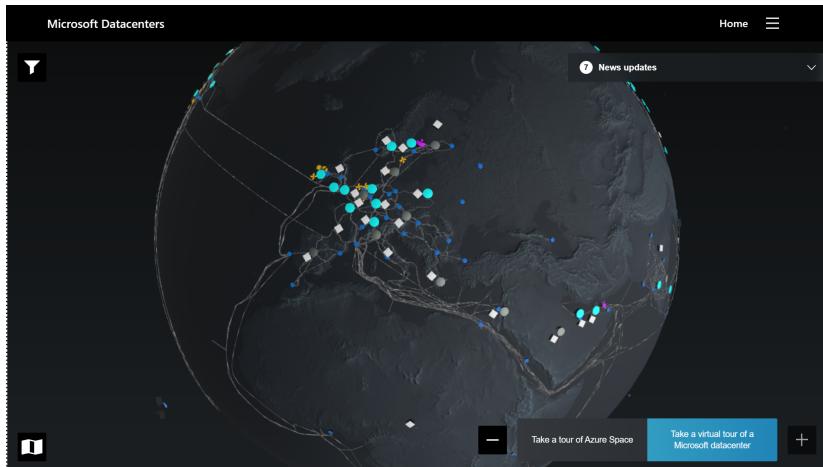
Źródło: [33]

Schemat 5.3 pokazuje jak obecnie podzielone są usługi oraz co jest udostępnione komercyjnie w ramach platformy Azure. Według schematu platforma podzielona jest na trzynaście obszarów, do których zaliczono między innymi bezpieczeństwo, zarządzanie danymi, usługi deweloperskie, analiza danych, platformy aplikacji.

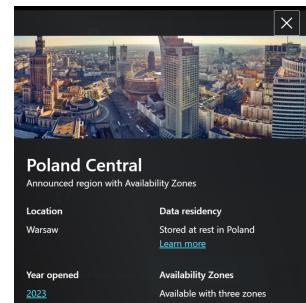
5.5. Infrastruktura

Infrastruktura globalna Azure składa się z dwóch części: fizycznej infrastruktury oraz globalnej łączności. Infrastruktura fizyczna składa się z ponad 200 centrów danych na całym świecie, połączonych w jedną globalną sieć. Takie rozwiązanie umożliwia wysoką skalowalność i dostępność poszczególnych rozwiązań. Cały ruch sieciowy jest utrzymywany wewnętrz prywatnej sieci Microsoft. Pozwala to na zachowanie informacji o adresach IP wewnętrz sieci, a co za tym idzie, informacje te nie trafiają do opinii publicznej [34].

Na swojej stronie internetowej Microsoft udostępnia wirtualną mapę, umożliwiającą zobaczenie aktualnej sieci Microsoftu oraz jej rozmieszczenie na globie ziemskim. Interaktywna mapa pozwala uzyskiwać informację o poszczególnych krajach oraz centrach danych znajdujących się na terytoriach tych krajów. Mapę oraz informację pokazano na **zdjęciach ??**.



(a) Globalna mapa infrastruktury sieciowej
Źródło: [35]



(b) Informacje o centrum danych
Źródło: [36]

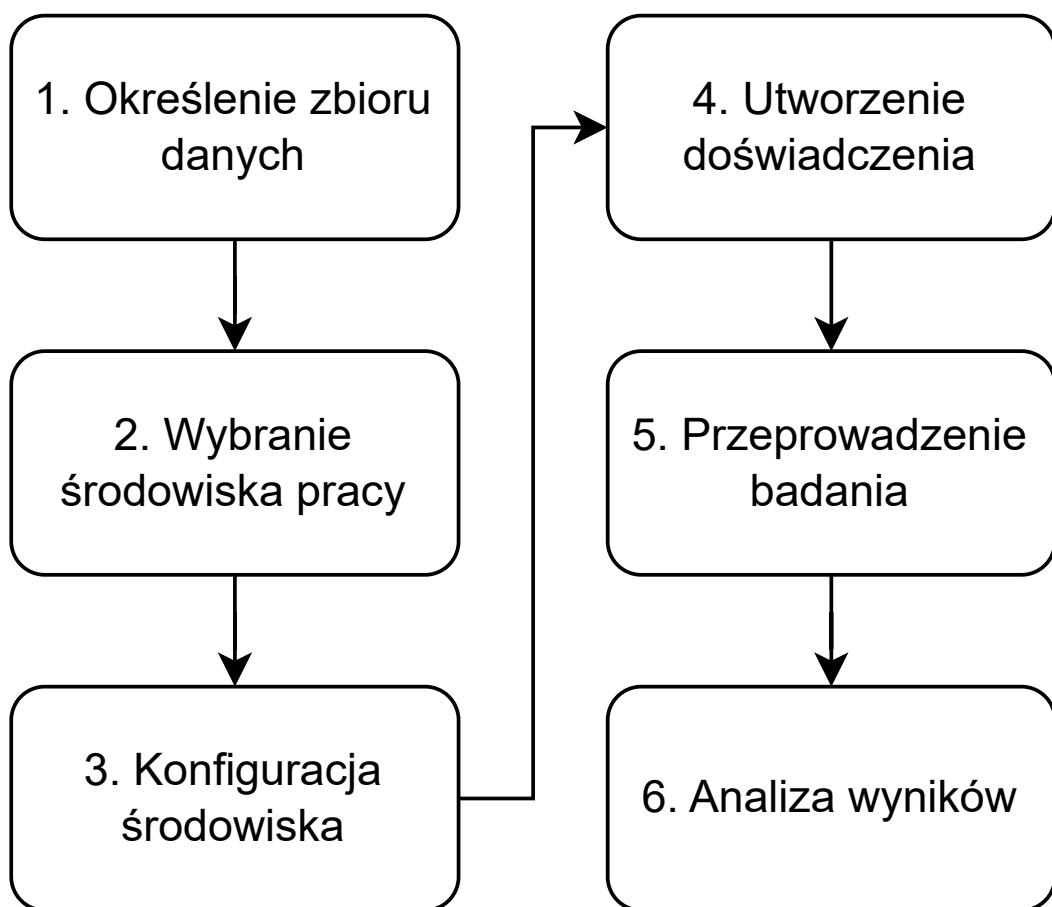
5.6. Machine Learning Studio

Azure Machine Learning Studio umożliwia łatwe i szybkie tworzenie wysoce wydajnych modeli uczenia maszynowego, a także zarządzanie nimi. Rozwiążanie wspiera pełen cykl życia kompleksowego uczenia maszynowego. Platforma umożliwia tworzenie potoków zadań, które połączone w jeden potok, wykonują poszczególne zadania w odpowiedniej kolejności. Dzięki modułowej budowie potoków uzyskano rozwiązywanie wielokrotnego użytku. W ramach jednego doświadczenia każdy moduł może być buforowany. Pozwala to na „zapamiętanie” wyniku poprzedniego uruchomienia i wykorzystanie go ponownie (o ile dane wejściowe albo konfiguracja nie uległy zmianie). Dodatkowo poza predefiniowanymi operacjami można wykorzystać moduły języka Python/R. Kolejną możliwością jest wytworzenie rozwiązania w technologii „**Jupyter Notebook**” oraz wizualne narzędzie wykorzystujące mechanizm *„przeciągnij i upuść”* (ang. *Drag & Drop*). Rozwiążanie to pozwala układać „*kafelki*” służące do tworzenia potoków zadań. Każde zadanie wykorzystuje wcześniej przygotowaną jednostkę obliczeniową, dzięki czemu można przewidzieć albo dostosować koszt wykorzystania modelu. Umożliwione zostało również wdrażanie modeli jako punktów końcowych. Pozwala to na komunikowanie się z nimi za pomocą REST API.

Microsoft umożliwia płatność jedynie za użytkowanie usług, co oznacza, że jeśli klaszter komputerowy był wykorzystywany jedynie przez 1 godzinę, to za tą jedną godzinę zostanie obciążony klient[37].

6. Opis doświadczenia

Przeprowadzone doświadczenie polega na porównaniu dostępnych w środowisku Microsoft Azure algorytmów klasyfikacji danych dwuklasowych wraz z algorytmem stworzonym na potrzeby pracy inżynierskiej o tytule „*Wykorzystanie algorytmów genetycznych w systemach wykrywania intruzów w sieciach komputerowych*” [1] oraz z algorymem DANet [38]. Doświadczenie przebiegało według **schematu 6.1**.



Rys. 6.1. Schemat przebiegu doświadczenia
Źródło: Opracowanie własne

6.1. Założenie techniczne

Dane prezentowane w **Tabeli 6.1** określają podstawowe założenia techniczne przyjęte w trakcie wykonywania analizy porównawczej. Dane te dotyczą między innymi środowiska, w którym wykonane było doświadczenie. Dodatkowo uwzględniono zestaw danych oraz biblioteki użyte w trakcie tworzenia doświadczenia.

Tabela 6.1. Założenia techniczne pracy dyplomowej
Źródło: Opracowanie własne

Środowisko uruchomieniowe	Machine Learning Studio[39]
Język programowania	Python 3.x
Wykorzystane biblioteki	scikit-learn [6]
	Numpy [40]
	Pandas [41, 42]
Wykorzystane dane	CICDS2017 [43]

6.2. Dane

Zbiór danych został przygotowany przez Kanadyjski Instytut Cyberbezpieczeństwa działający przy Uniwersytecie Nowy Brunszwik. Został wykonany za pomocą narzędzia CICFlowMeter [44]. Zbiór zawiera 79 cech ruchu sieciowego, do których zaliczyć można:

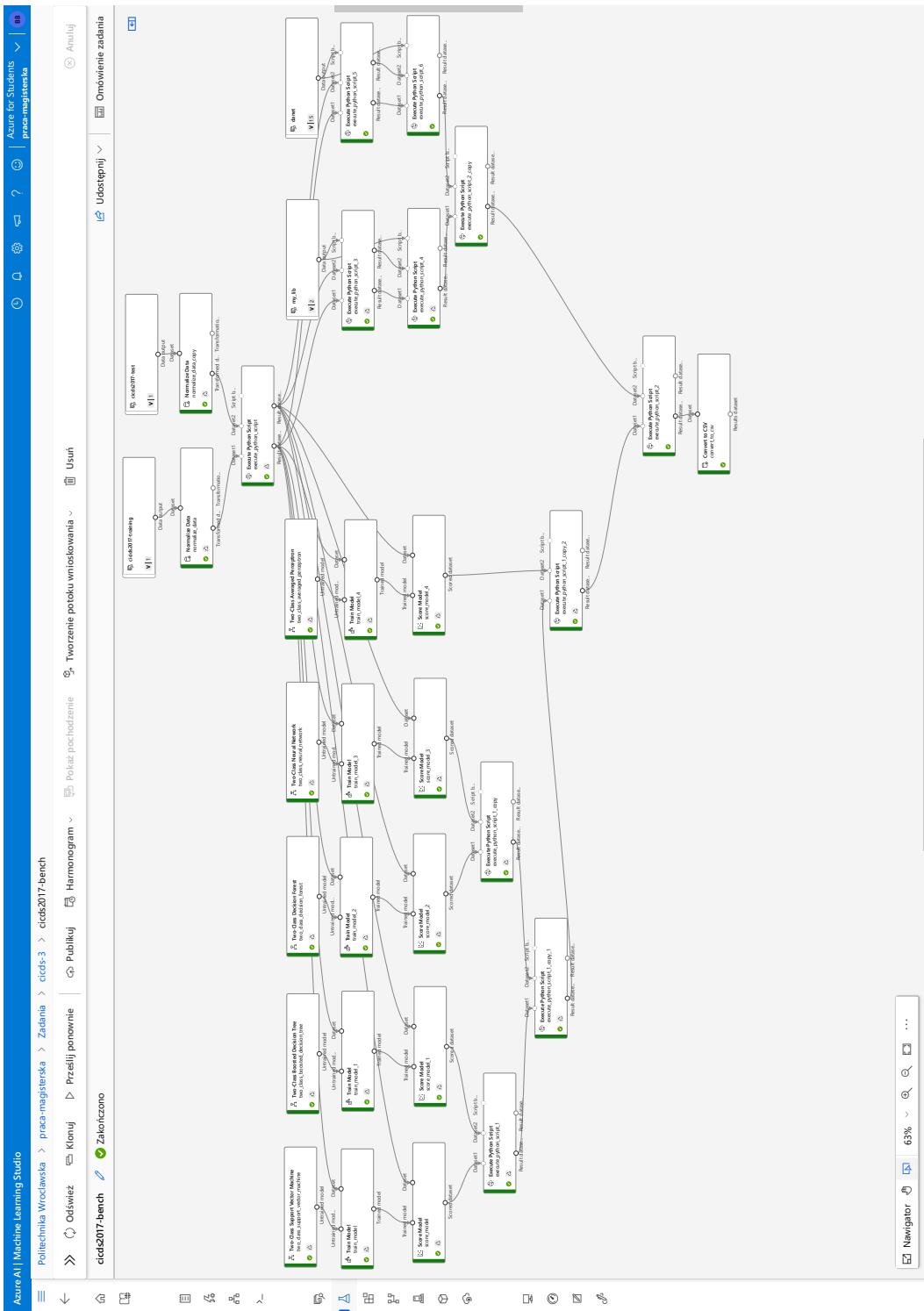
1. etykietę,
2. czas trwania przesyłu,
3. minimalną długość pakietu zwrotnego,
4. maksymalną długość pakietu zwrotnego,
5. port docelowy,
6. długość pakietów.

Zbiór pozwala na określenie czy ruch sieciowy jest życzliwy (*ang. BENING*), czy nieżyczliwy (różne możliwe formy ataku na sieć). Dodatkowo zbiór został podzielony na pięć dni roboczych: poniedziałek 3.07.2017 - piątek 7.07.2017. Dane z poniedziałku zawierają jedynie ruch życzliwy. W pozostałe dni zostały zasymulowane ataki na sieć komputerową [1, 45].

6.3. Programistyczne środowisko badawcze

Jako środowisko programistyczne zostało wybrane Azure Machine Learning Studio. Zostało to spowodowane możliwością uniezależnienia obliczeń od komputera lokalnego. Platforma umożliwia łatwy sposób na tworzenie skomplikowanych potoków zadań, które składają się z komponentów wielokrotnego użytku. Każdy komponent uruchamia się w środowisku odizolowanym od pozostałych operacji. Dzieje się tak dzięki wykorzystaniu wielowęzłowych klastrów obliczeniowych, bazujących na oprogramowaniu Docker. Klastry te mogą skalować się w zależności od potrzeb oraz dostępnej jednostki [46].

Całe doświadczenie zostało odwzorowane w graficznym potoku narzędzia „*Projektant*” oraz przedstawione na **zdjęciu 6.2**.



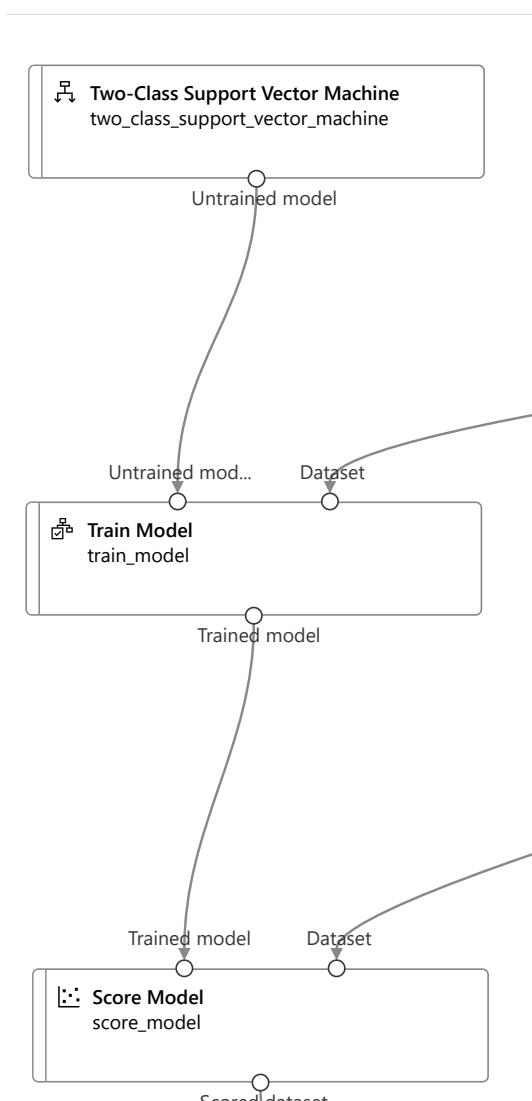
Rys. 6.2. Potok zadań
Źródło: Opracowanie własne

6.4. Algorytmy wykorzystane w doświadczeniu

W trakcie eksperymentu zastosowano różne algorytmy klasyfikacji danych. Charakterystyczną cechą tych algorytmów jest klasyfikacja ukierunkowana na 2 kategorie wejściowe. W tym wypadku są to kategorie ruchu sieciowego: [**BENIGN** (*pl. życzliwy*), **OTHER** (*pl. inne*)], gdzie inne to pozostałe typy ruchu sieciowego.

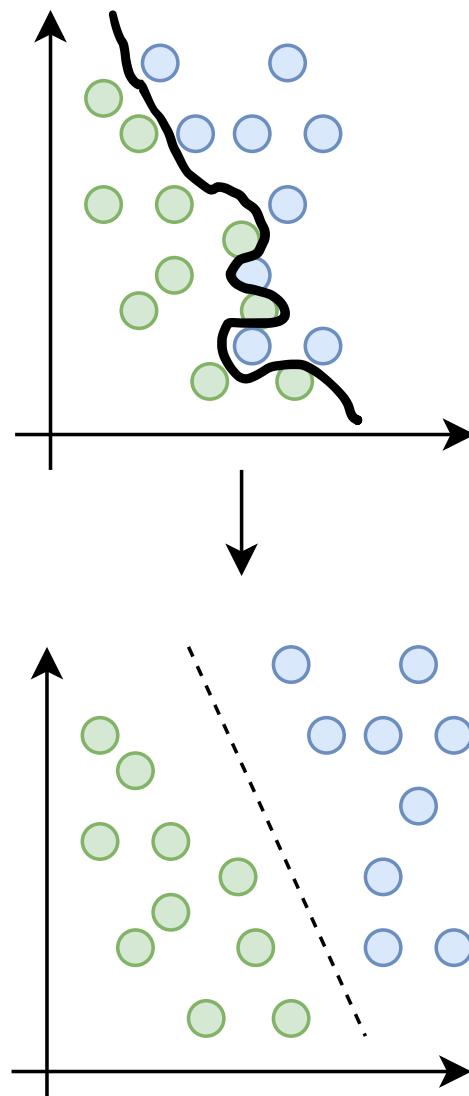
6.4.1. Two-Class Support Vector Machine

Algorytm SVM ma za zadanie znaleźć hiperpłaszczyznę w przestrzeni K-wymiarowej (K - liczba cech), która rozdziela zbiory punktów odpowiadających różnym klasom. W pierwszej kolejności szuka się separatora między klasami, a następnie przekształca się dane w taki sposób, by można przekształcić separator w hiperpłaszczyznę [47]. Sposób działania został zobrazowany za pomocą wykresów 6.3b. Część potoku 6.2 odpowiedzialnego za SVM to schemat 6.3a.



(a) Potok zadań dla modelu *Two-Class Support Vector Machine*

Zródło: Opracowanie własne

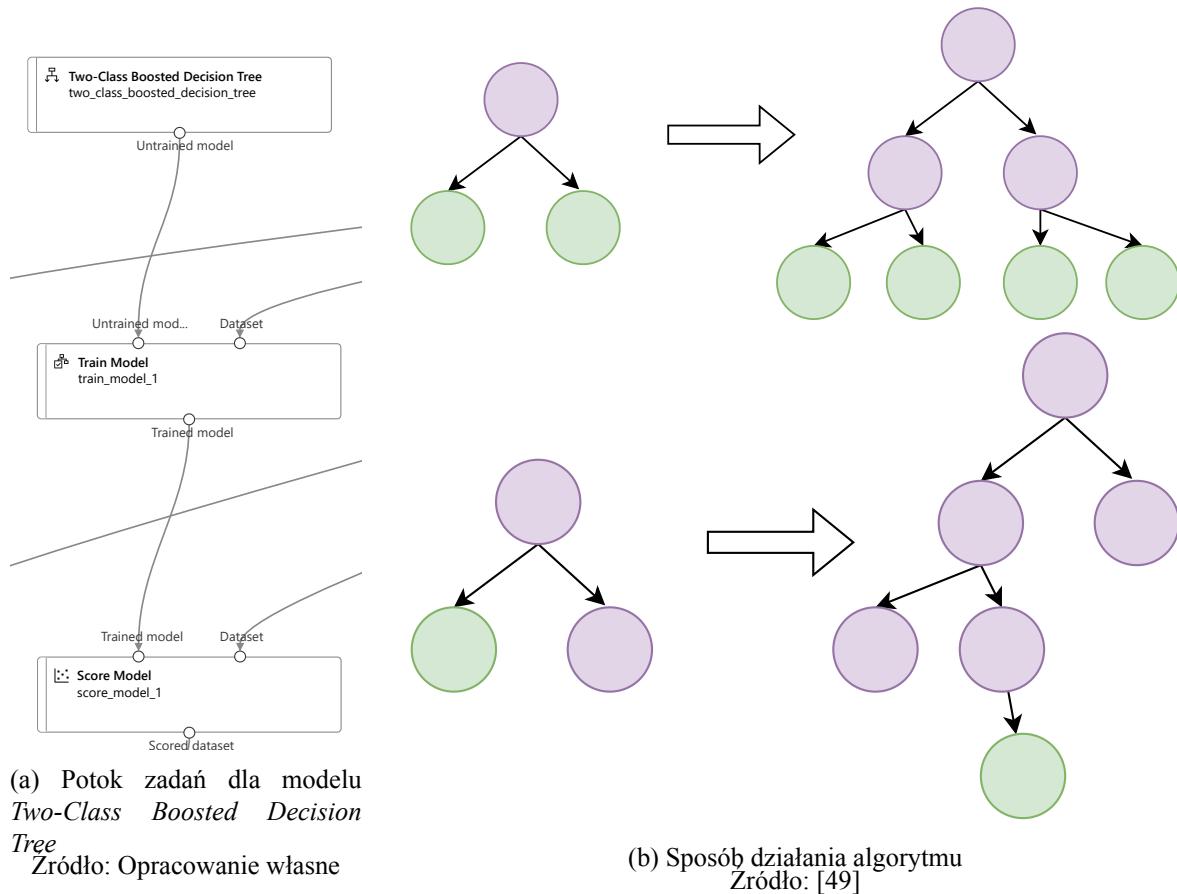


(b) Schemat SVM

Zródło: [48]

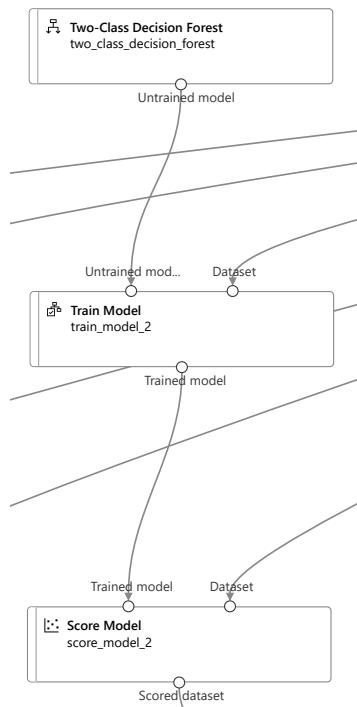
6.4.2. Two-Class Boosted Decision Tree

Jest to algorytm drzewa decyzyjnego oparty o algorytm LightGBM. Dzięki zastosowaniu tego podejścia algorytm oparty o drzewo decyzyjne działa szybciej oraz ma mniejszą złożoność obliczeniową. Algorytm ten działa na zasadzie doboru odpowiedniego liścia, zamiast jak w przypadku klasycznych algorytmów opartych na drzewie, wyboru odpowiedniej warstwy [49]. Sposób podejścia liściastego został ukazany na **schemacie 6.4b**. Model wykorzystywany w Azure ML został ukazany na **rysunku 6.4a**.



6.4.3. Two-Class Decision Forest

Las decyzyjny to algorytm, którego wynik opiera się o agregację wyników wielu drzew decyzyjnych. Uzyskanie wyniku zależy od algorytmu trenowania lasu. Przykładowo w klasifikacji losowym lasem wieloklasowym (*ang. Multi-class random forest classification*), każde drzewo głosuje na jedną klasę. Klasa, która zostanie wybrana większością głosów, zostaje uznana za wynikową [50]. Model wykorzystany w Azure ML pokazano na **modelu 6.5**

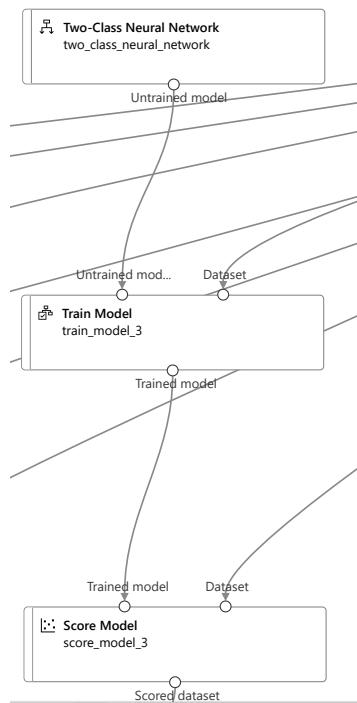


Rys. 6.5. Potok zadań dla modelu *Two-Class Decision Forest*

Źródło: Opracowanie własne

6.4.4. Two-class Neural Network

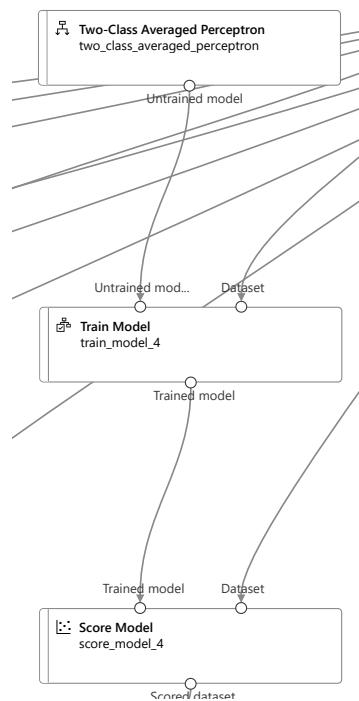
Jest to sieć neuronowa, która składa się z warstwy wejściowej, trzech warstw ukrytych (każda posiada po 100 węzłów), oraz z warstwy wyjściowej. Przykładowa sieć neuronowa została zobrazowana na **schemacie 3.6**. Moduł wykorzystany w Azure ML ukazano na **rysunku 6.6**.



Rys. 6.6. Potok zadań dla modelu *Two-Class Neural Network*
Źródło: Opracowanie własne

6.4.5. Two-Class Average Perceptron

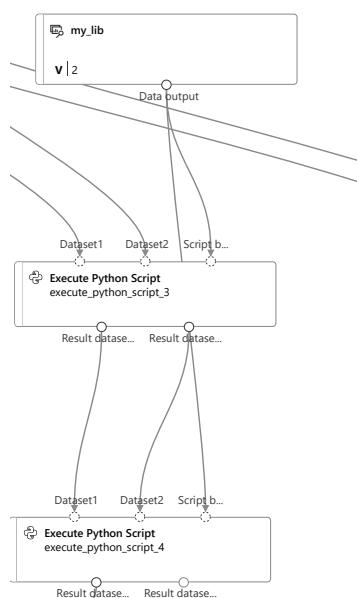
Jest to najprostsza odmiana sieci neuronowej, czyli pojedynczy perceptron, który jest matematycznym modelem neuronu. Składa się on z n wejść, takiej samej ilości wag, progu Θ , sumatora, funkcji aktywującej i wyjścia. Został zobrazowany na **schemacie 3.7**. Może służyć za prosty klasyfikator binarny albo za regresor. Model wykorzystany w Azure ML ukazano na **zdjęciu 6.7**.



Rys. 6.7. Potok zadań dla modelu *Two-Class Average Perceptron*
Źródło: Opracowanie własne

6.4.6. Gausian Naive Bayes - with GA

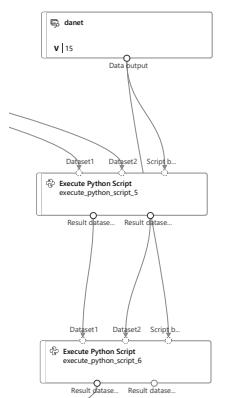
Algorytm ten polega na połączeniu algorytmu genetycznego (GA) wraz z klasyfikatorem naiwnym Bayesa wykorzystującego rozkład Gaussa (GNB). Zadaniem algorytmu genetycznego jest znalezienie najistotniejszych cech w zbiorze tabelarycznym. Poszukiwane cechy powinny pozwolić na zmniejszenie wymiarowości danych oraz na zmniejszenie kosztów obsługi samego klasyfikatora. Co może zostać uzyskane późniejszych etapach testowania, ze względu na zmniejszoną ilość danych wymaganych do przetworzenia. GA wykorzystywał w metodzie **fitness** algorytm GNB w celu określenia dopasowania danych. Zadaniem GNB było znalezienie najlepszej dostępnej kombinacji cech, które pozwalały na uzyskanie najlepszego dopasowania [1]. Model wykorzystywany w Azure ML różni się od gotowych modeli tym, że dołączono do niego bibliotekę napisaną w języku Python, która zawiera kod wykorzystywany w pracy inżynierskiej autora [51].



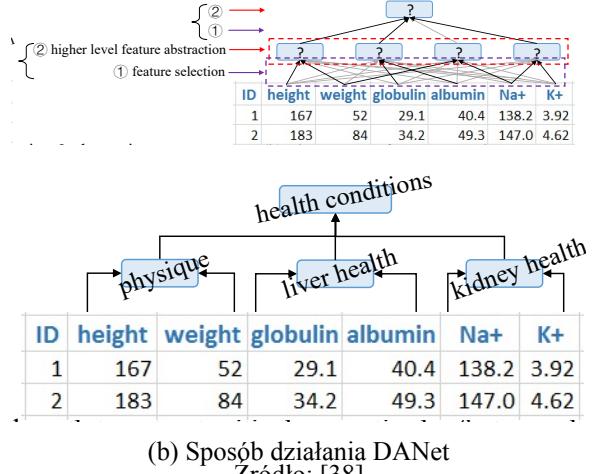
Rys. 6.8. Potok zadań dla modelu
Źródło: Opracowanie własne

6.4.7. DANet

Twórcy tego algorytmu wprowadzają dodatkową warstwę abstrakcyjną o nazwie „*Abstract Layer*”. Warstwy te budują sieć o nazwie „*Deep Abstract Network*” (DANet). Zadanie dodatkowych warstw jest grupowanie cech w skorelowanych zbiorach. Zbiory te budują sieć powiązań między sobą w formie sieci semantycznej. Gdy sieć semantyczna jest zbudowana, to w ostatnim kroku wykonywana jest klasyfikacja w trzywarstwowej sieci perceptronów (ang. *Multilayer Perceptron network*) [38, 52]. Model znajdujący się w Azure ML został przedstawiony na **zdjęciu 6.9a**, zaś sposób działania ukazano na **schematach 6.9b**.



(a) Potok zadań dla modelu *DANet*
Zródło: Opracowanie własne



7. Przebieg eksperymentu

Doświadczenie polegało na analizie porównawczej sprawności algorytmów opisanych w **rozdziale 6** w **sekcji 6.4**. Celem doświadczenia było określenie jakości algorytmu utworzonego w ramach pracy inżynierskiej autora [1]. Szczegółowa metodologia badawcza została określona w **podrozdziale 7.1**.

7.1. Metodologia badawcza

Przyjęta w projekcie metodologia badawcza została określona w poniżej **tabeli 7.1**. Przyjęta metodologia ma za zadanie określić jakość porównywanego algorytmu.

Tabela 7.1. Metodologia badawcza

Źródło: Opracowanie własne

Problem badawczy:

Czy algorytm klasyfikacji danych utworzony w ramach pracy inżynierskiej może konkurować z rozwiązaniami dostępnymi w środowiskach komercyjnych

Pytania badawcze:

1. Czy algorytm jest konkurencyjny pod względem wybranych metry:

- dokładność algorytmu
- czas działania
- precyzja
- czułość
- f1
- auc

Hipotezy:

1. Nie ma istotnej różnicy pomiędzy wynikami próby testowej i treningowej.
2. Nie ma istotnej różnicy pomiędzy wynikami prób testowych.
3. Wynik dopasowania algorytmów nie przekracza dolnej granicy przedziału ufności dla próby testowej

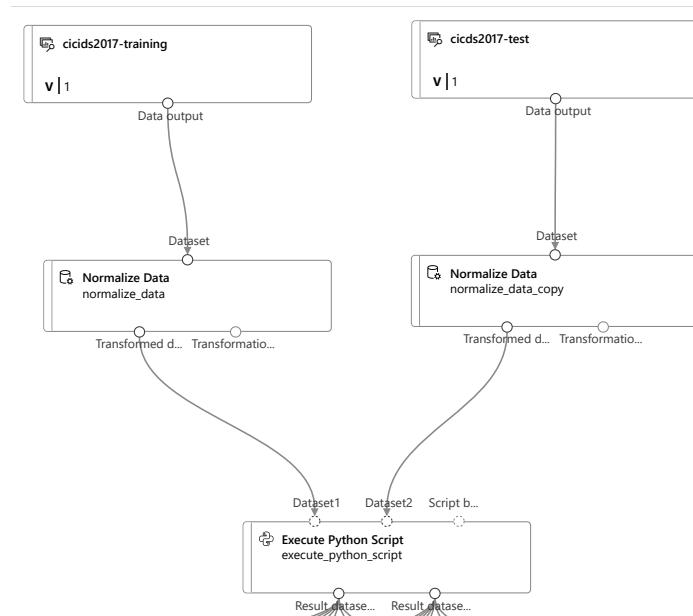
7.2. Przygotowanie platformy badawczej

Do badań wykorzystano narzędzie „*Projektant*” znajdujące się na platformie „*Azure Machine Learning Studio*” (Azure ML). Narzędzie to umożliwiło utworzenie interaktywnego potoku zadań. Potok ten składa się z kilku części:

- Przygotowanie i obróbka zbiorów danych
- Trenowanie oraz testowanie algorytmów klasyfikacji danych
- Utworzenie tabeli porównawczej dla wyników poszczególnych algorytmów (**obraz 6.2**).

7.2.1. Przygotowanie danych

W pierwszym kroku dane zostały znormalizowane za pomocą metody **MinMax**, która przekształca dane numeryczne do wartości w zakresie 0, 1. W kolejnym kroku za pomocą języka Python oraz bibliotek Pandas oraz Numpy zostają zamienione etykiety słowne na wartości **0** i **1** oraz następuje zamiana wartości $[NaN, -inf, inf]$ na cyfrę 0. Cały proces został zobrazowany na **diagramie 7.1**.



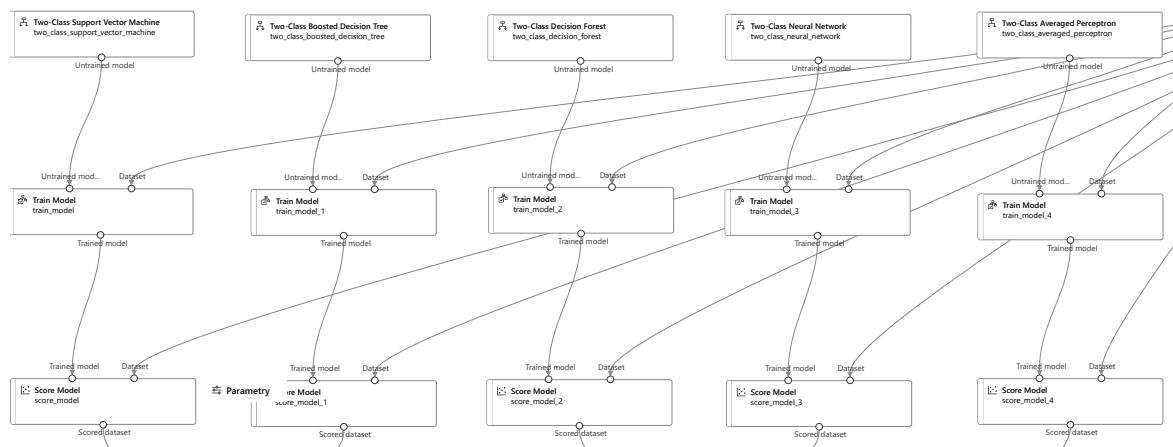
Rys. 7.1. Potok normalizacji danych
Źródło: Opracowanie własne

7.2.2. Trenowanie oraz testowanie algorytmów

Kolejną grupą zadań widoczną w potoku są te związane z trenowaniem i testowaniem poszczególnych algorytmów opisanych w **rozdziale 6**. Każdy test składa się 3 kafelek. W przypadku algorytmów dostarczonych wraz z platformą Azure ML są to:

- **model klasyfikujący** - odpowiada za przygotowanie algorytmu klasyfikacyjnego
- **blok treningowy** - tworzy wytrenowany model, za pomocą połączonego zbioru danych
- **blok ewaluacyjny** - sprawdza wcześniej wytrenowany model za pomocą powiązanego zbioru danych.

Potok zadań wykorzystujący algorytmy dostarczone przez Microsoft Azure został ukazany na **schemacie 7.2**

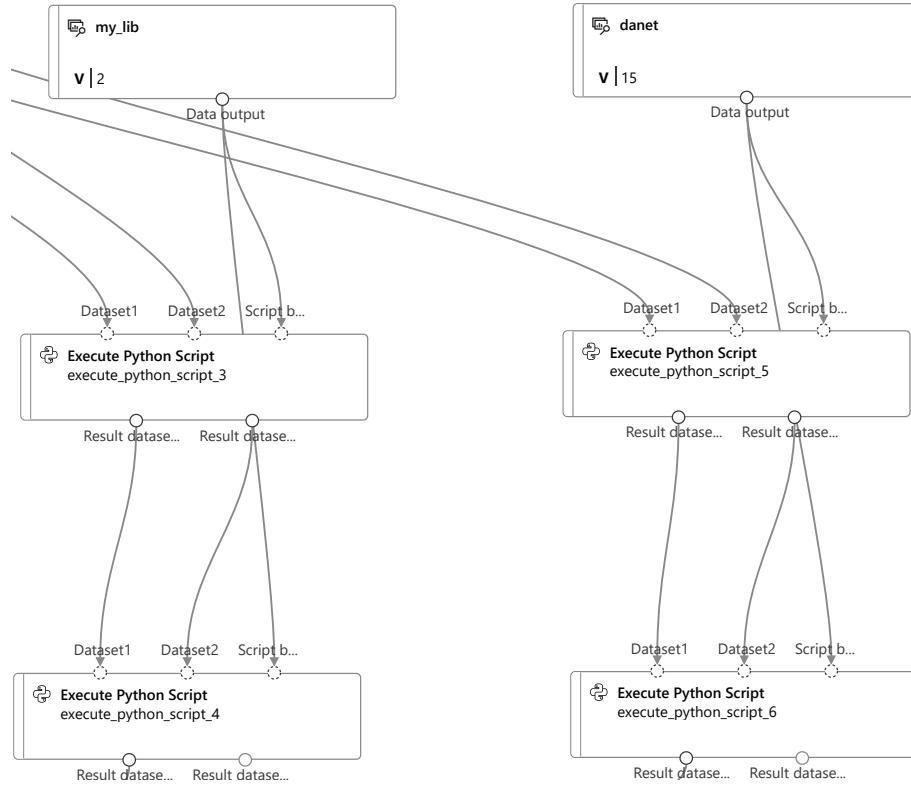


Rys. 7.2. Potok zadań dla algorytmów klasyfikacyjnych
Źródło: Opracowanie własne

Algorytmy dostarczone w ramach pracy badawczej składają się z:

- **biblioteka Python** - archiwum o rozszerzeniu **.zip**, które zawiera w sobie odpowiednie pliki napisane w języku Python
- **blok treningowy** - wykorzystuje dostarczoną bibliotekę do wytrenowania modelu oraz zapisania na platformie Azure najlepszego uzyskanego wyniku za pomocą powiązanego zbioru danych
- **blok ewaluacyjny** - wykorzystuje dostarczoną bibliotekę do ewaluacji algorytmu za pomocą połączonego zbioru danych

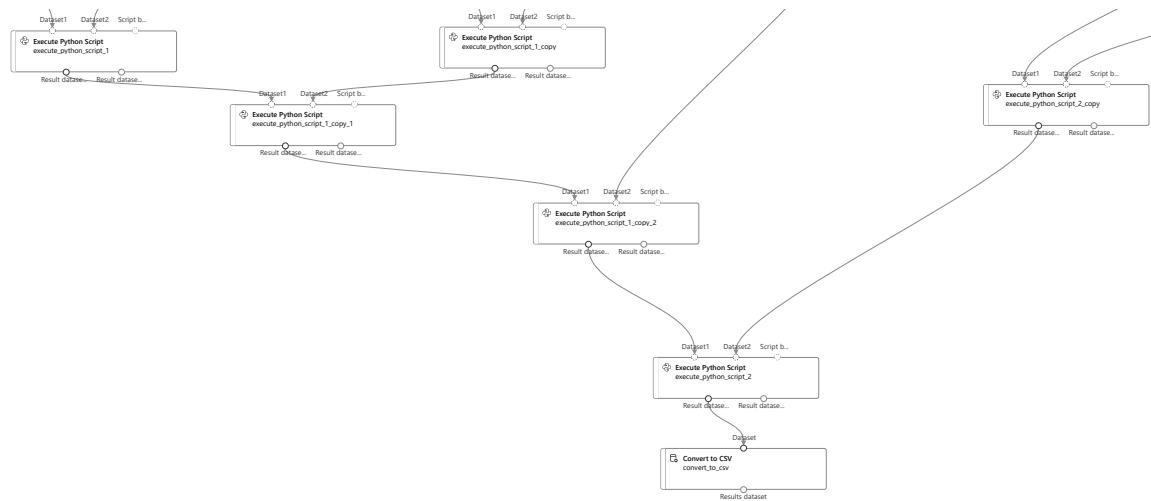
Potok zadań dla algorytmów niestandardowych został ukazany na **rysunku 7.3**



Rys. 7.3. Potok zadań dla algorytmów klasyfikacyjnych
Źródło: Opracowanie własne

7.2.3. Utworzenie tabeli porównawczej

Kolejną częścią zadań jest zebranie wyników poszczególnych algorytmów oraz połączenie ich w jedną całość. Wykorzystano do tego moduły języka Python, które zwracają przetworzone wyniki oraz łączą je w jedną tabelę zbiorczą, co pokazano na **rysunku 7.4.**



Rys. 7.4. Moduły odpowiedzialne za przetwarzanie wyników
Źródło: Opracowanie własne

7.3. Weryfikacja potoku

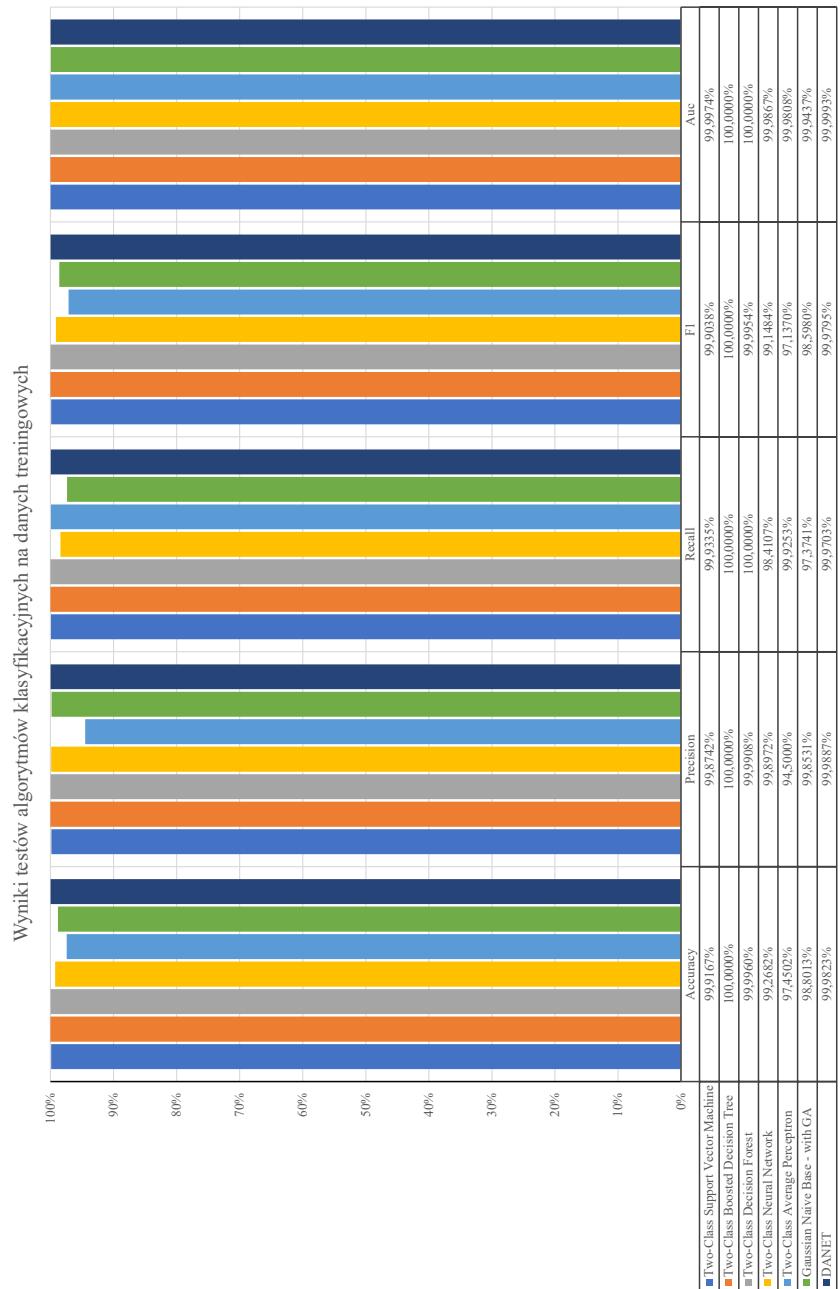
Aby zweryfikować działanie całego procesu wykorzystano znormalizowane dane treningowe do wytrenowania oraz przetestowania działania algorytmów klasyfikacyjnych. Cały proces trwał „**1 dzień 10 godzin 55 minut 53 sekundy**”. Wyniki tych działań widać na **rysunku 7.5**. Analizując wykres można zauważyć, że uzyskane wyniki znajdują się w przedziale [94%, 100%] w każdej metryce co pokazuje jakość każdego z algorytmów, a także to, że algorytmy poradziły sobie niemal bezbłędnie w rozpoznawaniu ruchu sieciowego, na którym były uczone. Zbiór, który wykorzystano do trenowania oraz testowania danych zawierał w sobie 225805 wpisów z czego 97718 należało do klasy „**1**”, zaś 128087 należało do klasy „**0**”.

Tabela 7.2. Liczba elementów przynależnych do danej klasy w zbiorze treningowym

Źródło: Opracowanie własne

Klasa	Liczba wystąpień
1	97718
0	128027
Suma	225805

Bazując na tym zbiorze oraz uzyskanych wynikach udało się udowodnić poprawność działania procesu klasyfikacji wieloma algorytmami genetycznymi.



Rys. 7.5. Wyniki testów algorytmów klasyfikacyjnych na danych treningowych
Źródło: Opracowanie własne

7.4. Próba badawcza

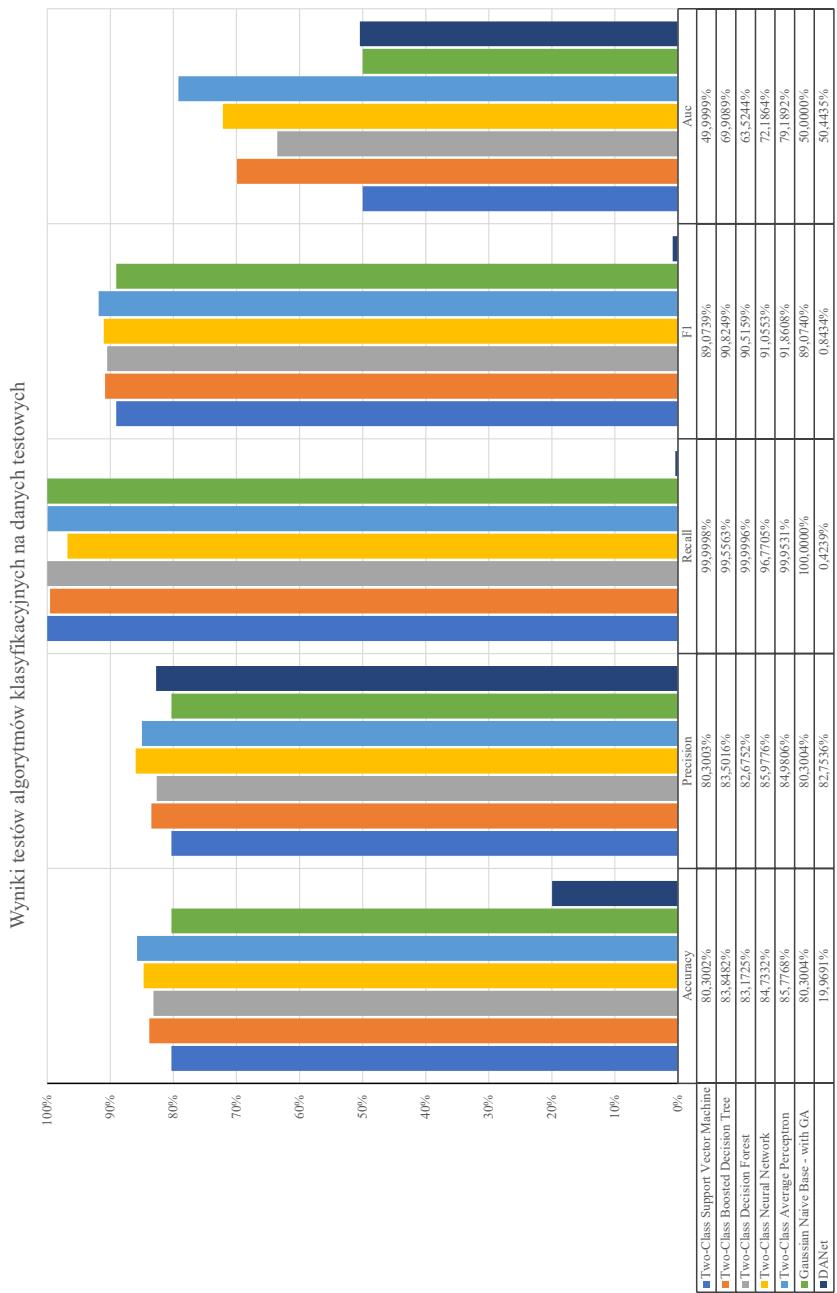
Aby uzyskać realne wyniki podczas porównywania poszczególnych algorytmów zastosowano zbiór treningowy opisany w **tabeli 7.2** oraz zbiór testowy, który zawierał 2273097 wpisów należących do klasy „1” oraz 557646 wpisów należących do klasy „0”. Sumarycznie ilość wpisów wynosi: 2830743, co zostało pokazane w **tabeli 7.3**. Pomiary testowe powtórzono 2 razy dzięki czemu uzyskano 3 próby badawcze.

Tabela 7.3. Liczba elementów przynależnych do danej klasy w zbiorze testowym

Źródło: Opracowanie własne

Klasa	Liczba wystąpień
1	2273097
0	557646
Suma	2830743

Poniżej zostały przedstawione wyniki zbiorcze dla poszczególnych metryk. Dodatkowo przedstawiono również wynik pomiaru treningowego, który w większości przypadków jest wyższy od danych testowych. Co prawdopodobnie jest spowodowane różnicą w ilości danych testowych i treningowych. Dodatkowo w każdej kolumnie oznaczono kolorem zielonym najwyższy wynik dla danej metryki, a kolorem czerwonym najniższy wynik dla danej metryki.



Rys. 7.6. Wyniki testów algorytmów klasyfikacyjnych na danych testowych
 Źródło: Opracowanie własne

7.4.1. Wyniki dopasowania

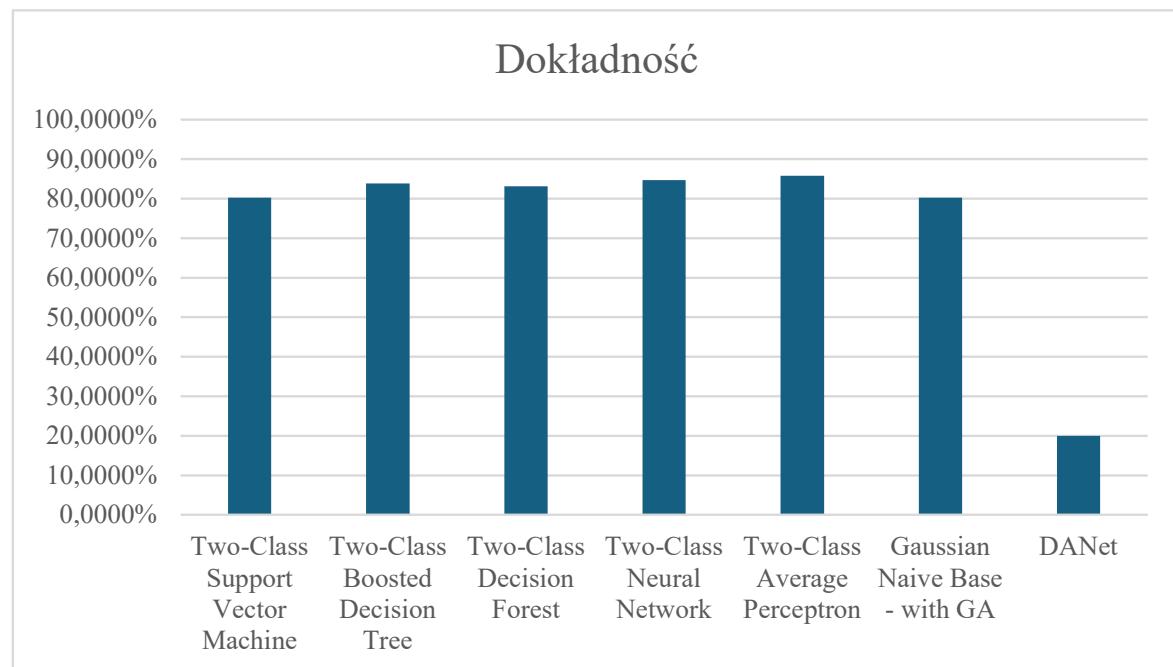
Najlepszy wynik dopasowania dla danych testowych uzyskał algorytm *Two-Class Average Perceptron*, który poprawnie rozpoznał 85,7768% próbek. Najgorszy wynik uzyskał algorytm *DANet* z dopasowaniem rzędu: 19,9691%. Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* z wynikiem 100,00%, a najgorszy *Two-Class Average Perceptron* z wynikiem 97,4502%. Wyniki dopasowania dla poszczególnych prób zostały przedstawione na **tabeli 7.4** oraz na **wykresie 7.7**.

Tabela 7.4. Wynik dopasowania algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik dopasowania			
	Próba 1	Próba 2	Próba 3	Próba testowa
Two-Class Support Vector Machine	80,3002%	80,3002%	80,3002%	99,9167%
Two-Class Boosted Decision Tree	83,8482%	83,8482%	83,8482%	100,0000%
Two-Class Decision Forest	83,1725%	83,1725%	83,1725%	99,9960%
Two-Class Neural Network	84,7332%	84,7332%	84,7332%	99,2682%
Two-Class Average Perceptron	85,7768%	85,7768%	85,7768%	97,4502%
Gaussian Naive Base - with GA	80,3004%	80,3004%	80,3004%	98,8013%
DANet	19,9691%	19,7899%	19,7899%	99,9823%



Rys. 7.7. Dokładność algorytmów
Źródło: Opracowanie własne

7.4.2. Wyniki precyzji

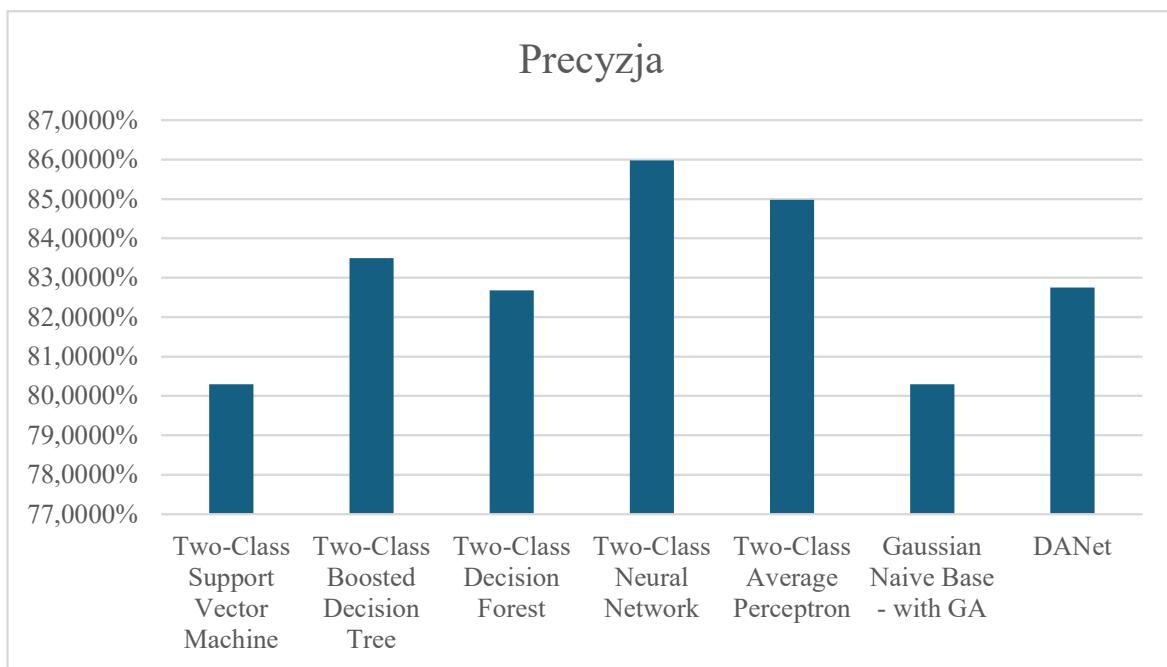
Najlepszy wynik precyzji dla danych testowych uzyskał algorytm *Two-Class Neural Network* (85, 9776%). Najgorszy wynik uzyskał algorytm *Two-Class Support Vector Machine* (80, 3003%). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* (100%), a najgorszy *Two-Class Average Perceptron* (94, 5%). Wyniki precyzji dla poszczególnych prób zostały przedstawione w **tabeli 7.5** oraz na wykresie **wykresie 7.8**.

Tabela 7.5. Wynik precyzji algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik precyzji			
	Próba 1	Próba 2	Próba 3	Próba testowa
Two-Class Support Vector Machine	80, 3003%	80, 3003%	80, 3003%	99, 8742%
Two-Class Boosted Decision Tree	83, 5016%	83, 5016%	83, 5016%	100, 0000%
Two-Class Decision Forest	82, 6752%	82, 6752%	82, 6752%	99, 9908%
Two-Class Neural Network	85, 9776%	85, 9776%	85, 9776%	99, 8972%
Two-Class Average Perceptron	84, 9806%	84, 9806%	84, 9806%	94, 5000%
Gaussian Naive Base - with GA	85, 7768%	80, 3004%	80, 3004%	99, 8531%
DANet	82, 7536%	85, 9516%	85, 9516%	99, 9887%



Rys. 7.8. Precyzja algorytmów

Źródło: Opracowanie własne

7.4.3. Wyniki czułości

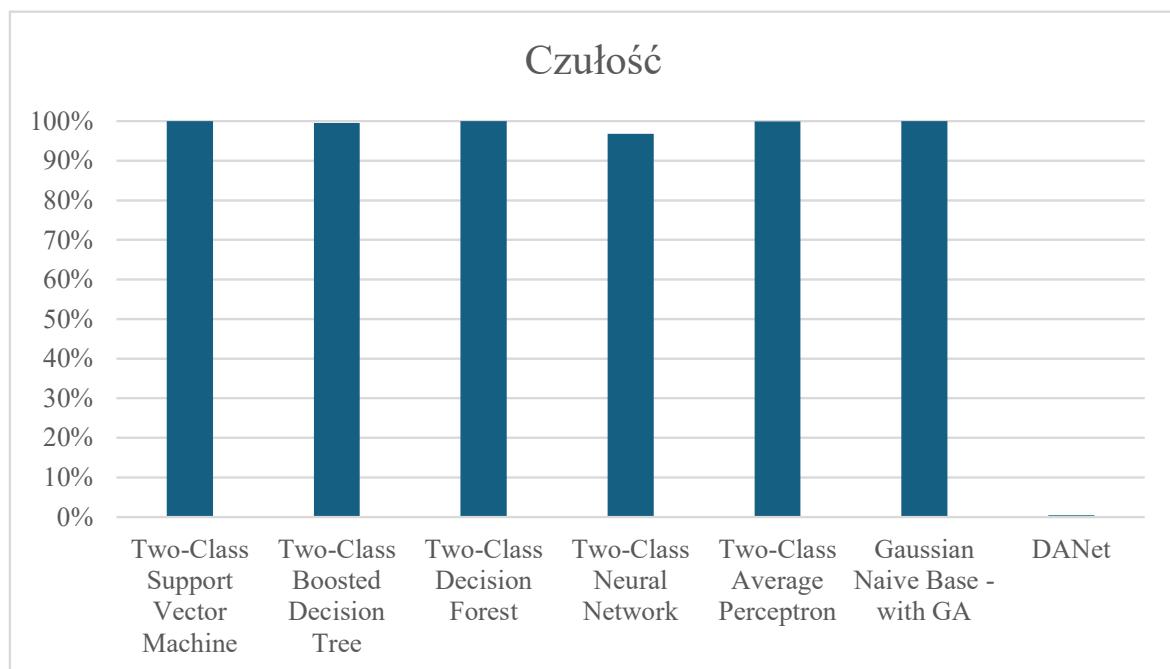
Najlepszy wynik czułości dla danych testowych uzyskał algorytm *Gaussian Naive Base - with GA* (100, 00%). Najgorszy wynik uzyskał algorytm *DANet* (0, 4239%). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* oraz *Two-Class Decision Forest* (100%), a najgorszy *Gaussian Naive Base - with GA* (97, 3741%). Wyniki czułości dla poszczególnych prób zostały przedstawione w **tabeli 7.6** oraz na wykresie **wykresie 7.9**.

Tabela 7.6. Wynik czułości algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik czułości			
	Próba 1	Próba 2	Próba 3	Próba testowa
Two-Class Support Vector Machine	99, 9998%	99, 9998%	99, 9998%	99, 9335%
Two-Class Boosted Decision Tree	99, 5563%	99, 5563%	99, 5563%	100, 0000%
Two-Class Decision Forest	99, 9996%	99, 9996%	99, 9996%	100, 0000%
Two-Class Neural Network	96, 7705%	96, 7705%	96, 7705%	98, 4107%
Two-Class Average Perceptron	99, 9531%	99, 9531%	99, 9531%	99, 9253%
Gaussian Naive Base - with GA	100, 0000%	100, 0000%	100, 0000%	97, 3741%
DANet	0, 4239%	0, 1343%	0, 1343%	99, 9703%



Rys. 7.9. Czułość algorytmów
Źródło: Opracowanie własne

7.4.4. Wyniki f1

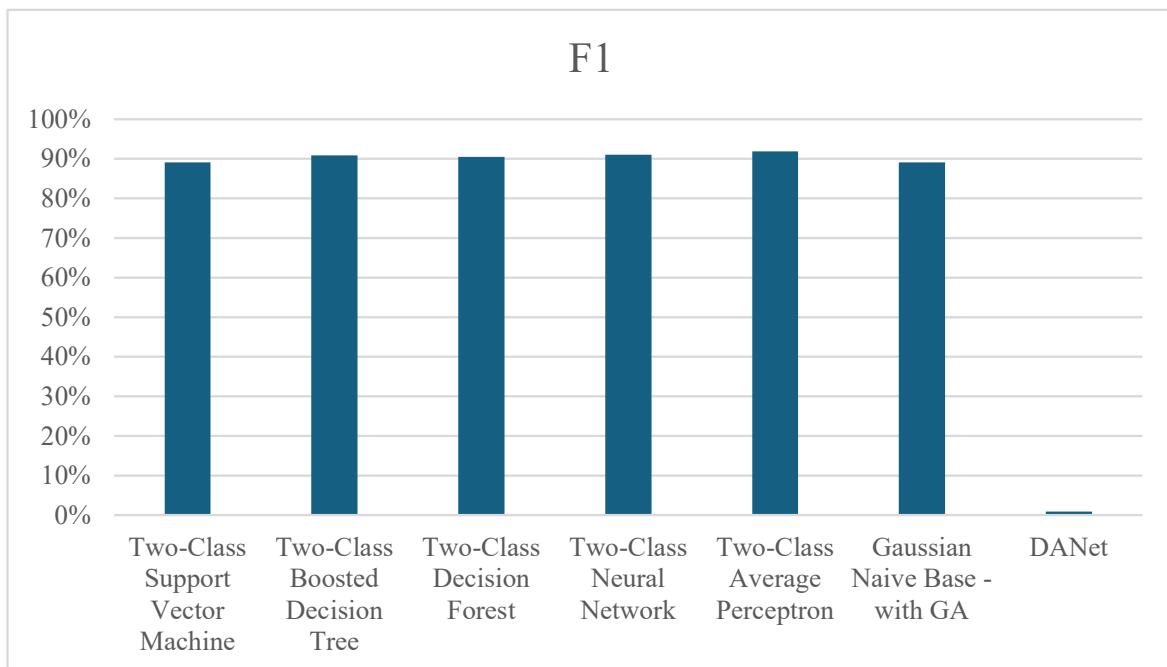
Najlepszy wynik F1 dla danych testowych uzyskał algorytm *Two-Class Average Perceptron* (91,8606%). Najgorszy wynik uzyskał algorytm *DANet* (0,8434%). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* (100%), a najgorszy *Two-Class Average Perceptron* (97,1370%). Wyniki precyzji dla poszczególnych prób zostały przedstawione w **tabeli 7.7** oraz na wykresie **wykresie 7.10**.

Tabela 7.7. Wynik F1 algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik F1			
	Próba 1	Próba 2	Próba 3	Próba testowa
Two-Class Support Vector Machine	89,0739%	89,0739%	89,0739%	99,9038%
Two-Class Boosted Decision Tree	90,8249%	90,8249%	90,8249%	100,0000%
Two-Class Decision Forest	90,5159%	90,5159%	90,5159%	99,9954%
Two-Class Neural Network	91,0553%	91,0553%	91,0553%	99,1484%
Two-Class Average Perceptron	91,8608%	91,8608%	91,8608%	97,1370%
Gaussian Naive Base - with GA	89,0740%	89,0740%	89,0740%	98,5980%
DANet	0,8434%	0,2682%	0,2682%	99,9795%



Rys. 7.10. F1 algorytmów
Źródło: Opracowanie własne

7.4.5. Wyniki AUC

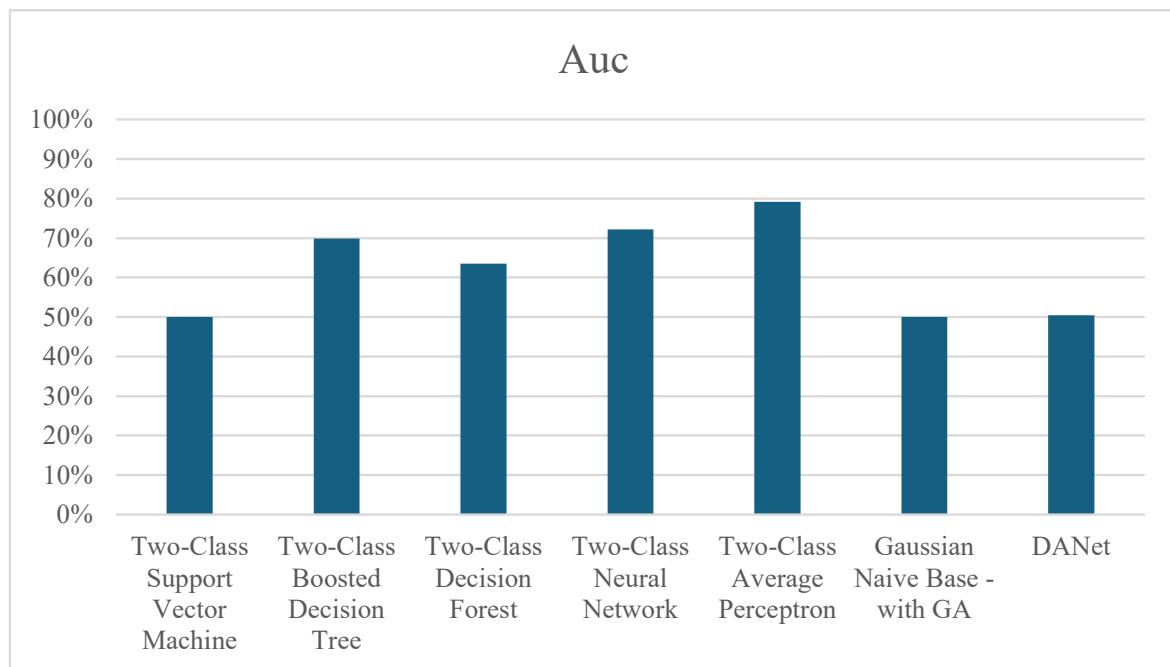
Najlepszy wynik precyzji dla danych testowych uzyskał algorytm *Two-Class AveragePerceptron* (79,1892%). Najgorszy wynik uzyskał algorytm *Two-Class Support Vector Machine* (49,999%). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* oraz *Two-Class Decision Forest* (100%), a najgorszy *Gaussian Naive Base - with GA* (99,9437%). Wyniki precyzji dla poszczególnych prób zostały przedstawione w **tabeli 7.8** oraz na wykresie **wykresie 7.11**.

Tabela 7.8. Wynik AUC algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik AUC			
	Próba 1	Próba 2	Próba 3	Próba testowa
Two-Class Support Vector Machine	49,9999%	49,9999%	49,9999%	99,9974%
Two-Class Boosted Decision Tree	69,9089%	69,9089%	69,9089%	100,0000%
Two-Class Decision Forest	63,5244%	63,5244%	63,5244%	100,0000%
Two-Class Neural Network	72,1864%	72,1864%	72,1864%	99,9867%
Two-Class Average Perceptron	79,1892%	79,1892%	79,1892%	99,9808%
Gaussian Naive Base - with GA	50,0000%	50,0000%	50,0000%	99,9437%
DANet	50,4435%	76,7282%	76,7282%	99,9993%



Rys. 7.11. AUC algorytmów
Źródło: Opracowanie własne

7.5. Analiza wyników

Wszystkie poniższe testy statystyczne zostały wykonane dla założeń z **tabeli 7.9**:

Tabela 7.9. Założenia wykorzystywane do analizy statystycznej danych

Źródło: Opracowanie własne

Założenie	Wartość
Przedział ufności	95%
α	0,05
Liczba elementów	7

- **Hipoteza H_0 : Nie ma istotnej różnicy pomiędzy wynikami "dopasowania" próby testowej i treningowej**

Wykorzystując statystyczny test t Studenta dla prób zależnych dla danych z **tabeli 7.4** oraz założenia z **tabeli 7.9** określono, że wartość $p-value = 0,0163$. Oznacza to, że zmienna $p-value < \alpha$, dzięki czemu można odrzucić hipotezę H_0 . Wyniki tej analizy określają, że widać istotne różnice pomiędzy danymi z próby testowej i treningowej. Największą różnicę widać w rezultacie algorytmu *DANet*, który uzyskał w próbie testowej 19,9691% dopasowania, a w próbie treningowej 99,9823%.

- **Hipoteza H_0 : Nie ma istotnej różnicy pomiędzy wynikami "dopasowania" prób testowych**

Z pomocą testu t Studenta dla prób zależnych określono porównano dane w próbach testowych z **tabeli 7.4**. Uzyskane wyniki, przedstawione w **tabeli 7.10**, pozwalają zachować Hipotezę H_0 , stwierdzającą, że pomiędzy danymi w poszczególnych próbach testowych nie ma istotnych różnic.

Tabela 7.10. Wyniki testu t Studenta dla poszczególnych prób testowych

Źródło: Opracowanie własne

H_0 Brak istotnych różnic między wynikami dla $\alpha = 0,05$		
Relacja	P-value	Rezultat
Próba 2 \leftrightarrow Próba 3	$0,5 > \alpha$	Brak istotnych różnic
Próba 2 \leftrightarrow Próba 1	$0,5 > \alpha$	Brak istotnych różnic
Próba 3 \leftrightarrow Próba 1	$0,5 > \alpha$	Brak istotnych różnic

- **Hipoteza H_0 : Wynik dopasowania algorytmów nie przekracza dolnej granicy przedziału ufności**

Przedział ufności dla dokładności algorytmów wyniósł 17,7218 dla $\alpha = 0.05$. Biorąc pod uwagę ten fakt można stwierdzić, że dokładność algorytmu *DANet* jest poniżej dolnej granicy. Dolna granica przedziału ufności wynosi 56,2925%, zaś *DANet* uzyskał 19,9691% oraz poprawnie zaklasyfikował jedynie 565273 wpisów. Oznacza to, że można odrzucić H_0 , ponieważ jeden algorytm przekracza dolny próg granicy ufności

7.6. Wnioski

Microsoft Wyszedł naprzeciw potrzebom użytkowników przygotowując zestaw prekonfigurowanych algorytmów klasyfikacyjnych. Możliwości narzędzia Azure ML pozwalają na odpowiadanie na konkretne potrzeby przy relatywnie niewielkich kosztach. To nie oznacza jednak, że tworzenie autorskich rozwiązań mija się z celem. Jak ukazano na **wykresie 7.6** autorskie rozwiązania również mają rację bytu. Wykorzystanie połączenia algorytmu genetycznego i klasyfikatora naiwnego Bayesa z rozkładem normalnym pozwala na uzyskanie zbliżonych wyników do algorytmu utworzonego przez giganta technologicznego. Różnica około 5 punktów procentowych między najlepszym algorytmem a algorytmem GAGNB ukazuje niewielką różnicę w jakości algorytmu. Dodatkowo wciąż trudno korzystać z rozwiązań takich jak DANet, które są nieprzetestowane na innych zbiorach niż tych przygotowanych z algorytmem.

Dodatkowo korzystanie z tego typu prostych rozwiązań autorskich pozwala na prototypownie rozwiązań biznesowych opartych o klasyfikację danych. Samo wykorzystanie algorytmu genetycznego z algorytmem GNB umożliwia skupienie się na tworzenie ogólnego rozwiązania. Nie wymaga to wcześniejszej znajomości zbioru oraz pozwala na korzystanie z programów klasyfikacyjnych lokalnie nie ponosząc kosztów wykorzystania platformy chmurowej. Kolejnym atutem utworzonego przez autora rozwiązania jest zmniejszenie kosztów lokalnego użytkowania. Co zostało spowodowane zmniejszeniem wymiarowości zbioru danych do klasyfikacji poprzez wykorzystanie jedynie wytypowanych kolumn.

Doświadczenie to ukazuje, że wciąż należy próbować tworzyć wydajniejsze i dokładniejsze rozwiązania, lecz nie zaprzecza faktu iż rozwiązania ogólnie dostępne są na bardzo wysokim poziomie. Uzyskano dokładność z zakresu [80%, 85%] przy czym plik testowy był 12 krotnie większy od pliku treningowego.

8. Perspektywy rozwoju

Stworzony projekt jest jedynie silnikiem klasyfikacyjnym, który pozwala na wytrenowanie i wyłonienie najlepszego algorytmu do klasyfikacji danych. Dzięki możliwościom platformy Azure ML stworzenie całego środowiska testowego jest relatywnie tanie i nie wymaga wielu wyspecjalizowanych umiejętności. Bazując na wynikach i najlepszym klasyfikatorze, można utworzyć odpowiednie przepływy służące do klasyfikacji danych wejściowych. Dostęp do nich może odbywać się za pomocą utworzonych Punktów Dostępowych (*ang. Endpoints*). Dzięki punktom dostępowym możliwa jest komunikacja za pomocą protokołu HTTPS (*ang. Hyper Text Transfer Protocol Secure*) i komunikacja typu REST (*ang. Representative State Transfer*).

Utworzony w ten sposób punkt dostępu może zostać wykorzystany w klasyfikacji ruchu sieciowego w niewielkich aplikacjach z dostępem do internetu. Pozwoliłoby to na realizację analizy danych w chmurze, co mogłoby przyspieszyć cały proces oraz utworzyć pojedyncze źródło prawdy dla wielu instancji aplikacji. A wykorzystanie dodatkowo konteneryzacji, którą zapewnia platforma Azure, cały proces mógłby zostać zoptymalizowany pod kątem wydajnościowym i lokalizacyjnym. Pojedyncze źródło prawdy jest zaletą wykorzystania „zewnętrznego” klasyfikatora, ponieważ zapewnia jednakowe wyniki klasyfikacji w poszczególnych instancjach samej aplikacji instalowanej na wielu urządzeniach. Pozwala to również na lepsze dostrajanie całego procesu, a wykorzystanie kolejnych wersji przepływów umożliwia utrzymywanie kopii zapasowych poszczególnych rozwiązań. Umożliwia to na przykład cofnięcie wersji klasyfikatora w przypadku wykrycia nieprawidłowości w obecnym modelu.

Wykaz rysunków

2.1 Schemat algorytmu genetycznego	9
2.2 Klasyfikacja zbioru danych: Monday-WorkingHours	12
3.1 Graficzne przedstawienie podziałów sztucznej inteligencji	13
3.2 Podział uczenia maszynowego	14
3.3 Uczenie nienadzorowane	14
3.4 Uczenie nadzorowane	15
3.5 Uczenie przez wzmacnianie	16
3.6 Schemat sieci neuronowej	17
3.7 Schemat pojedynczego neuronu - perceptronu	18
3.8 Schemat prostej głębszej sieci neuronowej	19
5.1 PowerApps od Microsoft	22
5.2 Wordpress.com	23
5.3 Schemat podziału usług MS Azure	26
6.1 Schemat przebiegu doświadczenia	28
6.2 Potok zadań	30
6.5 Potok zadań dla modelu <i>Two-Class Decision Forest</i>	33
6.6 Potok zadań dla modelu <i>Two-Class Neural Network</i>	34
6.7 Potok zadań dla modelu <i>Two-Class Average Perceptron</i>	35
6.8 Potok zadań dla modelu	36
7.1 Potok normalizacji danych	39
7.2 Potok zadań dla algorytmów klasyfikacyjnych	40
7.3 Potok zadań dla algorytmów klasyfikacyjnych	41
7.4 Moduły odpowiedzialne za przetwarzanie wyników	41
7.5 Wyniki testów algorytmów klasyfikacyjnych na danych treningowych	43
7.6 Wyniki testów algorytmów klasyfikacyjnych na danych testowych	45
7.7 Dokładność algorytmów	46
7.8 Precyzyja algorytmów	47
7.9 Czułość algorytmów	48
7.10 F1 algorytmów	49
7.11 AUC algorytmów	50

Wykaz tabel

2.1	Klasyfikacja zbioru danych: Monday-WorkingHours	11
4.1	Macierz pomyłek	20
6.1	Założenia techniczne pracy dyplomowej	29
7.1	Metodologia badawcza	38
7.2	Liczba elementów przynależnych do danej klasy w zbiorze treningowym .	42
7.3	Liczba elementów przynależnych do danej klasy w zbiorze testowym . .	44
7.4	Wynik dopasowania algorytmów.	46
7.5	Wynik precyzji algorytmów.	47
7.6	Wynik czułości algorytmów.	48
7.7	Wynik F1 algorytmów.	49
7.8	Wynik AUC algorytmów.	50
7.9	Założenia wykorzystywane do analizy statystycznej danych	51
7.10	Wyniki testu t Studenta dla poszczególnych prób testowych	51

Bibliografia

- [1] Bartosz Błyszcz. „*Wykorzystanie algorytmów genetycznych w systemach wykrywania intruzów w sieciach komputerowych*”. Praca Inżynierska. Kraków: Akademia Górnictwo-Hutnicza im. Stanisława Staszica w Krakowie, wrz. 2022.
- [2] Jan Kusiak, Anna Danielewska-Tulecka i Piotr Oprocha. „*Optymalizacja*”. Wydawnictwo Naukowe PWN SA, 2021. ISBN: 9788301159610.
- [3] Kumara Sastry, David Goldberg i Graham Kendall. „*Genetic algorithms*”. W: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, Boston, MA, 2005, s. 97–125. ISBN: 0387234608. doi: 10.1007/0-387-28356-0_4.
- [4] Joyce i James. „*Bayes’ Theorem (Stanford Encyclopedia of Philosophy/Spring 2019 Edition)*”. 2003. URL: <https://plato.stanford.edu/archives/spr2019/entries/bayes-theorem/> (term. wiz. 2022-05-12).
- [5] K Ming Leung. „*Naive bayesian classifier*”. 2007. URL: <http://cis.poly.edu/%7B~%7Dmleung/FRE7851/f07/naiveBayesianClassifier.pdf> (term. wiz. 2022-05-12).
- [6] F. Pedregosa i in. „*Scikit-learn: Machine Learning in Python*”. W: *Journal of Machine Learning Research* 12 (2011), s. 2825–2830.
- [7] Oxford University Press. „*intelligence, n., sense I*”. W: *Oxford English Dictionary*. Lip. 2023. doi: 10.1093/OED/3757635879.
- [8] „*Artificial Intelligence in Science*”. OECD, czer. 2023. doi: 10.1787/a8d820bd-en.
- [9] Batta Mahesh. „*Machine Learning Algorithms-A Review*”. W: *International Journal of Science and Research* (2018). ISSN: 2319-7064. doi: 10.21275/ART20203995.
- [10] Satavisa Pati. „*The Difference Between Artificial Intelligence and Machine Learning*”. W: *Emerj Ml* (2018), s. 3–8.
- [11] Chun Zhang i in. „*Semi-supervised behavioral learning and its application*”. W: *Optik* 127.1 (2016), s. 376–382. ISSN: 00304026. doi: 10.1016/j.ijleo.2015.10.089.
- [12] Sun-Chong Wang. „*Artificial Neural Network*”. W: *Interdisciplinary Computing in Java Programming* (2003), s. 81–100. doi: 10.1007/978-1-4615-0377-4_5.
- [13] Austin Pollard. „*What are neural networks?*” 1990. doi: 10.1108/eb007822. URL: <https://www.ibm.com/topics/neural-networks>.
- [14] Karolina Bartos. „*SIEĆ SOM JAKO PRZYKŁAD SIECI SAMOORGANIZUJĄCEJ SIĘ*”. W: (2012). ISSN: 1507-3866.
- [15] Microsoft. „*Deep learning vs. machine learning - Azure Machine Learning*”. 2022. URL: <https://learn.microsoft.com/en-us/azure/machine-learning/concept-deep-learning-vs-machine-learning?view=azureml-api-2> (term. wiz. 2023-09-02).
- [16] Algolytics. „*Jak ocenić jakość i poprawność modeli klasyfikacyjnych ? Część 4- Krzywa ROC*”. URL: <https://algolytics.pl/tutorial-jak-ocenic-jakosc-i-poprawnosc-modeli-klasyfikacyjnych-czesc-4-krzywa-roc/> (term. wiz. 2023-09-04).

- [17] Wordpress. „*WordPress.com: Build a Site, Sell Your Stuff*”. 2023. URL: <https://wordpress.com/> (term. wiz. 2023-09-04).
- [18] JoomlaORG. „*Joomla! - Content Management System to build websites*”. 2021. URL: <https://www.joomla.org/%20https://www.joomla.org/about-joomla.html> (term. wiz. 2023-09-04).
- [19] Wix. „*Free website builder | Create a free website*”. 2016. URL: <https://www.wix.com/> (term. wiz. 2023-09-04).
- [20] Microsoft. „*PowerApps*”. 2023. URL: <https://guidedtour.microsoft.com/guidedtour/scenarios/power-apps/2.2.png> (term. wiz. 2023-09-04).
- [21] Wordpress. „*Playground Demo*”. 2023. URL: <https://developer.wordpress.org/playground/demo/> (term. wiz. 2023-09-04).
- [22] Alexander C. Bock i Ulrich Frank. „*Low-Code Platform*”. W: *Business and Information Systems Engineering* 63.6 (grud. 2021), s. 733–740. ISSN: 18670202. DOI: [10.1007/S12599-021-00726-8/FIGURES/1](https://doi.org/10.1007/S12599-021-00726-8/FIGURES/1).
- [23] Martin Hirzel. „*Low-Code Programming Models*”. W: (maj 2022). arXiv: 2205.02282.
- [24] Microsoft. „*Co to jest usługa Power Apps? - Power Apps | Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/powerapps-overview> (term. wiz. 2023-09-05).
- [25] Microsoft. „*Omówienie tworzenia aplikacji kanw - Power Apps | Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/maker/canvas-apps/getting-started> (term. wiz. 2023-09-05).
- [26] Microsoft. „*Omówienie tworzenia aplikacji opartej na modelu z Power Apps - Power Apps | Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/maker/model-driven-apps/model-driven-app-overview> (term. wiz. 2023-09-05).
- [27] Microsoft. „*Omówienie kart dla usługi Power Apps - Power Apps | Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/cards/overview> (term. wiz. 2023-09-05).
- [28] AmazonQuickSight. „*Business Intelligence Service – Amazon QuickSight – AWS*”. URL: <https://aws.amazon.com/quicksight/> (term. wiz. 2023-09-05).
- [29] GoogleAppSheet. „*Google AppSheet | Build apps with no code*”. URL: <https://about.appspot.com/home/> (term. wiz. 2023-09-05).
- [30] Abandy Roosevelt. „*History of Microsoft Azure*”. 2022. URL: [https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204#](https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204%20https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204#) (term. wiz. 2023-09-08).
- [31] Microsoft Azure. „*Poznaj platformę Azure*”. URL: <https://azure.microsoft.com/pl-pl/explore/> (term. wiz. 2023-09-06).
- [32] Datashift. „*Microsoft Azure*”. URL: <https://www.datashift.eu/technology/microsoft-azure/?fbclid=IwAR262r9Kdc0oeFII8PmCmCuu-P6-5VSHnoKfPyjTJTsEmOkIgmVmfsuuiS8> (term. wiz. 2023-09-08).
- [33] Datashift. „*MS Azure.png*”. URL: <https://cdn.nimbu.io/s/znvdo1j/pages/8g7p2fo/MS%20Azure.png?33zmiw4> (term. wiz. 2023-09-08).

- [34] Microsoft Azure. „*Infrastruktura globalna*”. URL: <https://azure.microsoft.com/pl-pl/explore/global-infrastructure/> (term. wiz. 2023-09-06).
- [35] Microsoft Azure. „*Azure global infrastructure experience*”. URL: <https://datacenters.microsoft.com/globe/explore> (term. wiz. 2023-09-08).
- [36] Microsoft Azure. „*Azure global infrastructure experience*”. URL: https://datacenters.microsoft.com/globe/explore?info=region_polandcentral (term. wiz. 2023-09-08).
- [37] Microsoft Azure. „*Azure Machine Learning — uczenie maszynowe jako usługa*”. URL: <https://azure.microsoft.com/pl-pl/products/machine-learning> (term. wiz. 2023-09-08).
- [38] Jintai Chen i in. „*DANETs: Deep Abstract Networks for Tabular Data Classification and Regression*”. W: *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*. T. 36. Association for the Advancement of Artificial Intelligence, grud. 2022, s. 3930–3938. ISBN: 1577358767. doi: [10.1609/aaai.v36i4.20309](https://doi.org/10.1609/aaai.v36i4.20309). arXiv: [2112.02962](https://arxiv.org/abs/2112.02962).
- [39] Microsoft. „*Microsoft Machine Learning Studio (classic)*”. 2022. URL: <https://studio.azureml.net/> (term. wiz. 2023-09-01).
- [40] Charles R Harris i in. „*Array programming with NumPy*”. W: *Nature* 584 (7824 wrz. 2019), s. 356–362. doi: [9.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2).
- [41] The Pandas development team. „*pandas-dev/pandas: Pandas*”. Lut. 2019. doi: [9.5281/zenodo.3509134](https://doi.org/10.5281/zenodo.3509134). URL: <https://doi.org/10.5281/zenodo.3509134>.
- [42] Wes McKinney. „*Data Structures for Statistical Computing in Python*”. W: *Proceedings of the 9th Python in Science Conference*. 2010, s. 56–61. doi: [10.25080/majora-92bf1922-00a](https://doi.org/10.25080/majora-92bf1922-00a).
- [43] UNB. „*CICIDS2017 | Kaggle*”. URL: <https://www.kaggle.com/datasets/cicdataset/cicids2017> (term. wiz. 2023-09-01).
- [44] Ahlashkari. „*GitHub - ahlashkari/CICFlowMeter: CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is an Ethernet traffic Bi-flow generator and analyzer for anomaly detection that has been used in many Cybersecurity datasets such as Android Adware-General Malware datas*”. 2022. URL: <https://github.com/ahlashkari/CICFlowMeter> (term. wiz. 2023-09-11).
- [45] Iman Sharafaldin, Arash Habibi Lashkai i Ali A Ghorbani. „*IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*”. 2018. URL: <https://www.unb.ca/cic/datasets/ids-2017.html> (term. wiz. 2023-09-01).
- [46] Microsoft Learn. „*Create compute clusters - Azure Machine Learning | Microsoft Learn*”. 2023. URL: <https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-attach-compute-cluster?view=azureml-api-2&tabs=python> (term. wiz. 2023-09-11).
- [47] IBM. „*Sposób działania algorytmu SVM - IBM Documentation*”. URL: <https://www.ibm.com/docs/pl/spss-modeler/saas?topic=models-how-svm-works> (term. wiz. 2023-09-13).
- [48] Statsoft. „*SVMIntro3.gif(obraz GIF, 358×131 pikseli)*”. URL: <https://www.statsoft.pl/textbook/graphics/SVMIntro3.gif> (term. wiz. 2023-09-13).
- [49] LightGBM. „*Features — LightGBM 4.0.0 documentation*”. URL: <https://lightgbm.readthedocs.io/en/stable/Features.html> (term. wiz. 2023-09-16).

- [50] Google. „*Lazy decyzyjne | Machine Learning | Google for Developers*”. URL: <https://developers.google.com/machine-learning/decision-forests/intro-to-decision-forests-real?hl=pl> (term. wiz. 2023-09-16).
- [51] Suvres2023. „*GitHub - Suvres/gnb-gp: comparison of GNB with GNB-GA*”. URL: <https://github.com/Suvres/gnb-gp> (term. wiz. 2023-09-15).
- [52] Danet. „*GitHub - WhatAShot/DANet: DANets (a family of neural networks) for tabular data classification/ regression.*” URL: <https://github.com/WhatAShot/DANet> (term. wiz. 2023-09-01).